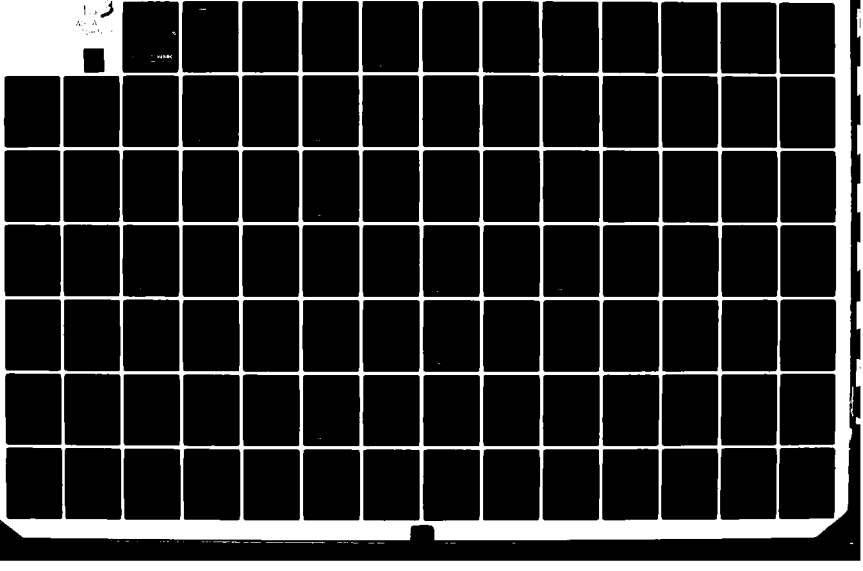
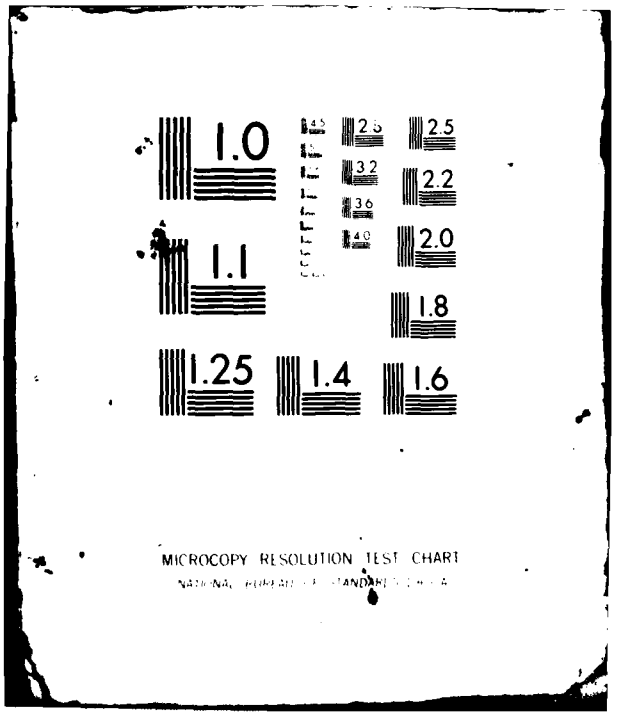
AD-A108 589 VERAC INC SAN DIEGO CA F/G 15/3.1

AN ANALYSIS OF MMCS NETWORK ARCHITECTURES TO SUPPORT THE DATA P--ETC(U)

DEC 80 J C TIERNAN DAS660-80-C-0017

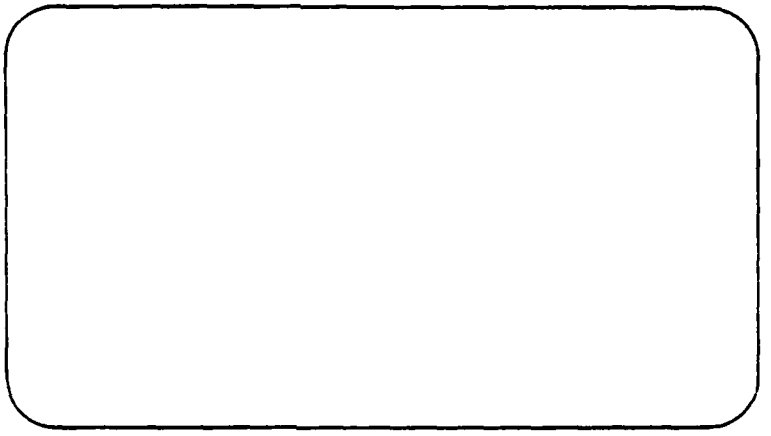
UNCLASSIFIED R-008-80 NL




1.0
1.1
1.25
1.4
1.6
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART

LEVEL II

AD A108589



DTIC
ELECTE
DEC 15 1981
S D
H

DTIC FILE COPY

VERAC incorporated

4901 Morena Blvd. Suite 209, San Diego, CA 92117 (714) 272-1360

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

81 12 14 118

VERAC incorporated

(1)

AN ANALYSIS OF

MMCS NETWORK ARCHITECTURES

TO SUPPORT THE

DATA PROCESSING REQUIREMENTS FOR

SITE DEFENSE

R-008-80

16 December 1980

Submitted to:

Commander
Ballistic Missile Defense Systems Command
Attention: BMDSC-HUL
DODA Code: W 31 RPD
PO Box 1500
Huntsville, Alabama 35807

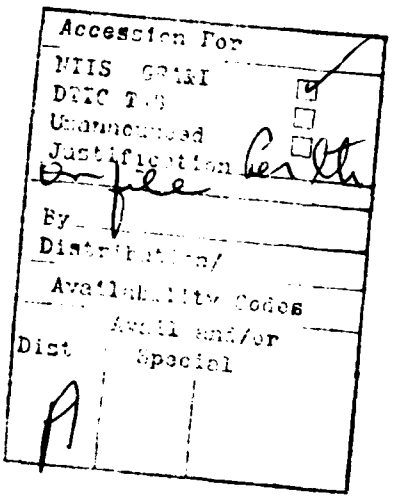
Prepared by: Harold J. Payne and: James C. Tiernan

Approved by: Charles L. Morefield

CONTRACT DASG60-80-C-0017

4901 Morena Boulevard, Suite 209
San Diego, California 92117
714/272-1360

PRELIMINARY NOTES



PREFACE

This document is provided to the U.S. Army Ballistic Missile Defense (BMD) Systems Technology Program (STP) Office in fulfillment of part of the documentation requirements associated with the Analysis of Alternative Distributed Network Architectures, a supporting effort for the Advanced Data Processing Subsystem Investigations (ADPSI).

The purpose of this document is to present the findings of this study, to indicate the expected critical design constraints imposed by capabilities of computer and communications hardware to be available in 1981-82, and to identify critical computer subsystem design issues. Further, the Processing Architecture Evaluation Methodology (PAEM) and the supporting computer simulation tool, Processing Architecture Evaluation Simulation (PAES) which were developed to support architecture design and analysis are described here.

TABLE OF CONTENTS

TABLE OF CONTENTS
(concluded)

LIST OF FIGURES

LIST OF FIGURES
(concluded)

LIST OF TABLES

EXECUTIVE SUMMARY

The U.S. Army Ballistic Missile Defense (BMD) Systems Technology Program (STP) office supports a program to maintain and update technologies critical to the deployment of a BMD system. One part of that program, the Advanced Data Processing Subsystem Investigation (ADPSI), has been directed toward the identification of key technical issues relating to the data processing subsystem, and the resolution of those technical issues that would be time-critical in a full scale development.

As a part of the ADPSI effort, VERAC, Inc. has conducted a study of alternative network architectures for a distributed data processing (DDP) approach to implementation of the BMD Site Defense data processing subsystem. This report is the Final Report documenting the activities and conclusions reached in the study.

The purpose of the VERAC study has been to develop and evaluate alternative distributed processing architectures for the BMD Site Defense Application. A number of architectures were to be assessed in this evaluation. These architectures include: (1) an advanced computer with a Centralized control and memory Architecture; (2) a Thread Architecture under study by McDonnel Douglas; and (3) a Hierarchical Architecture. VERAC has developed a model of the functional process supported by the Site Defense processor in order to perform the evaluation of alternative architectures. A perferred architecture has been identified.

PAEM/PAES

In order to meet the technical objectives of this study, VERAC extended a previously developed simulation tool for network flow analysis into a wholly new Processing Architecture Evaluation Methodology (PAEM), with a supporting tool, the Processing Architecture Evaluation Simulation (PAES). PAEM/PAES provides a highly disciplined and extremely flexible approach to characterization, design and analysis of centralized and distributed data processing systems. This approach will have great utility beyond this study, and in particular can support future BMD work such as the LoAD program.

Developments in computer technology have introduced a new direction in the design of dedicated data processing subsystems. With the impending availability of processing elements with substantial power and therefore applicability of DDP for Terminal Defense missions, the architectural design associated with the network of distributed computers now is on a level with the software design.

The Processing Architecture Evaluation Methodology and the supporting simulation tool, PAES, supports two stages of the DDP HW/SW design process. The first stage is the definition of the process in terms of the function to be supported, the environment (i.e., threat scenario) and the requirements. PAEM supports this by providing a formalized framework for representing the process. Further, the simulation tool, PAES, provides an effective means of checking the consistency of the functional process definition by exercising the functional model in a realistic model of the dynamic environment. The flexibility of PAES is a particularly important asset at this stage, since a variety of process models and threat scenarios are easily introduced for analysis.

PAEM/PAES can provide invaluable support to the second stage of design in which functions are allocated to hardware elements and the architectural characteristics of the DDP system are specified. PAEM provides a framework for specifying architectural elements (including

gross computer characteristics, operating systems, bus protocols and data base management techniques) in a fashion that pinpoints critical design issues. PAES provides an evaluation of a specific HW/SW design as it would be expected to function in a realistic, dynamic environment.

PAEM/PAES fills an important gap in otherwise available analysis tools. Analytic tools (e.g., static, queueing network models) and static simulations, while being flexible and efficient to use, cannot reflect the critical congestion effects which actually occur in the dynamic environment. The presently available tools which can provide useful measures of performance in the dynamic environment are complex simulations which are quite inflexible - requiring recoding to examine substantial deviations in the process definition or in the DDP architecture. These tools are also expensive to run. Thus, the ability of these complex simulations to provide an effective HW/SW design aid is severely limited. They are, however, valuable for final validation of performance of a design because of their fidelity in performance prediction.

PAES provides a dynamic simulation, yet retains flexibility in application by depending upon a table-driven simulator whose table elements constitute the definition of the process, scenario and architecture. Run time efficiency derives from the flow approximation which is traded against performance prediction accuracy by the user. As a result, PAES provides a means for rapidly assessing an alternative design, and can, in fact, be integrated as an important aid in the actual design process.

PAES itself is structured into three functional components (illustrated in Figure ES-1.):

(1) Input Specification - The process, architecture, scenario, and output reports required are separately specified. Automated verification of structural consistency is provided by this component of PAES.

(2) Simulation - A deterministic, flow-oriented representation of the processing of "work-units" operates on tables defined by the Input Specification component. Process and architectural effects are separately simulated. This provides the basis for architecture-free process simulation. The overall modularity makes additions of new architectural effects modules relatively easy. This component generates files as directed by the Report Specification.

(3) Report Generation - A User-interactive component produces performance tables and plots from files generated by the Simulation component.

The PAEM/PAES methodology has been applied to the BMD Site Defense processor as summarized in the following.

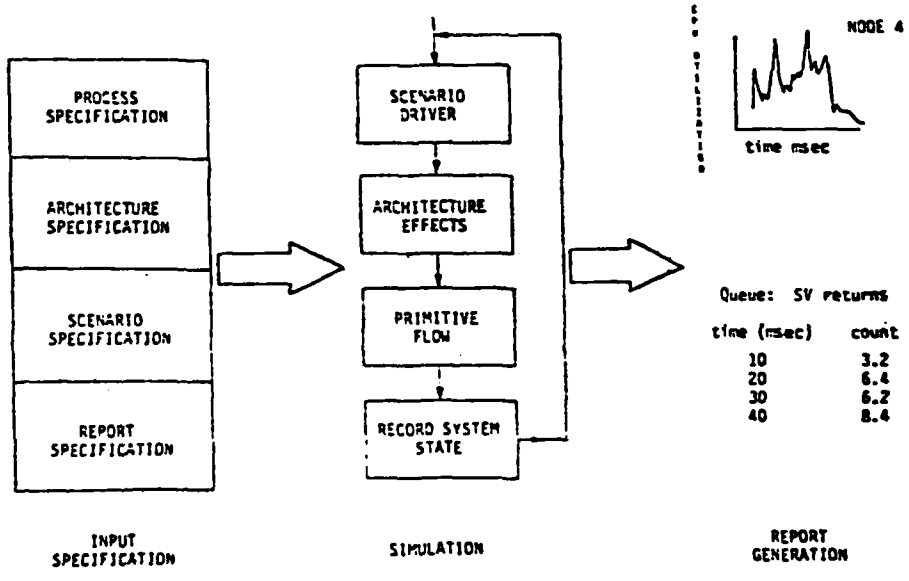


Figure ES-1. PAES - Processing Architecture Evaluation Simulation

SITE DEFENSE PROCESS DEFINITION AND ANALYSIS

Existing software designs have been used in addressing the functions to be supported. These designs are tied to the use of the CDC 7700 computer; as a result, many design aspects reflect CDC 7700-specific implementation considerations rather than the process supported by the implementation. A prime example of this is the partitioning of the software into tasks which collectively can be effectively scheduled by TOS, the Tactical Operating System.

VERAC first undertook an effort to obtain a machine-independent description of the BMD process. An overview of this process is presented in Figure ES-2. This effort was an essential first step in addressing DDP architecture design, and further, has yielded an extremely useful characterization for subsequent related analyses, and, ultimately, for design of the software on a DDP system.

This effort included the following steps: (1) an analysis of the existing BMD task designs, particularly those which were actually realized as code (radar interface, target and object tracking, and discrimination), to obtain a characterization of task code structure (functions, principal data structure, and timing models); (2) an aggregation of tasks based upon associations necessary for principal functional requirements (e.g., tasks on a processing thread were generally put together); (3) a sequential decomposition of these functons which ultimately yielded a set of PRIMITIVES, each of which is a smallest, logically cohesive functional process unit. These PRIMITIVES served as the basic units allocated in development of a distributed processing architecture.

Performance requirements and design constraints have been obtained from available documents. The principal requirements which have significant impact upon architecture are (1) the "port-to-port" response times associated with the processing of the various types of radar returns, and (2) the throughput or total computer processing load necessary to support all functions in selected scenarios. These requirements served to guide the Process Definition.

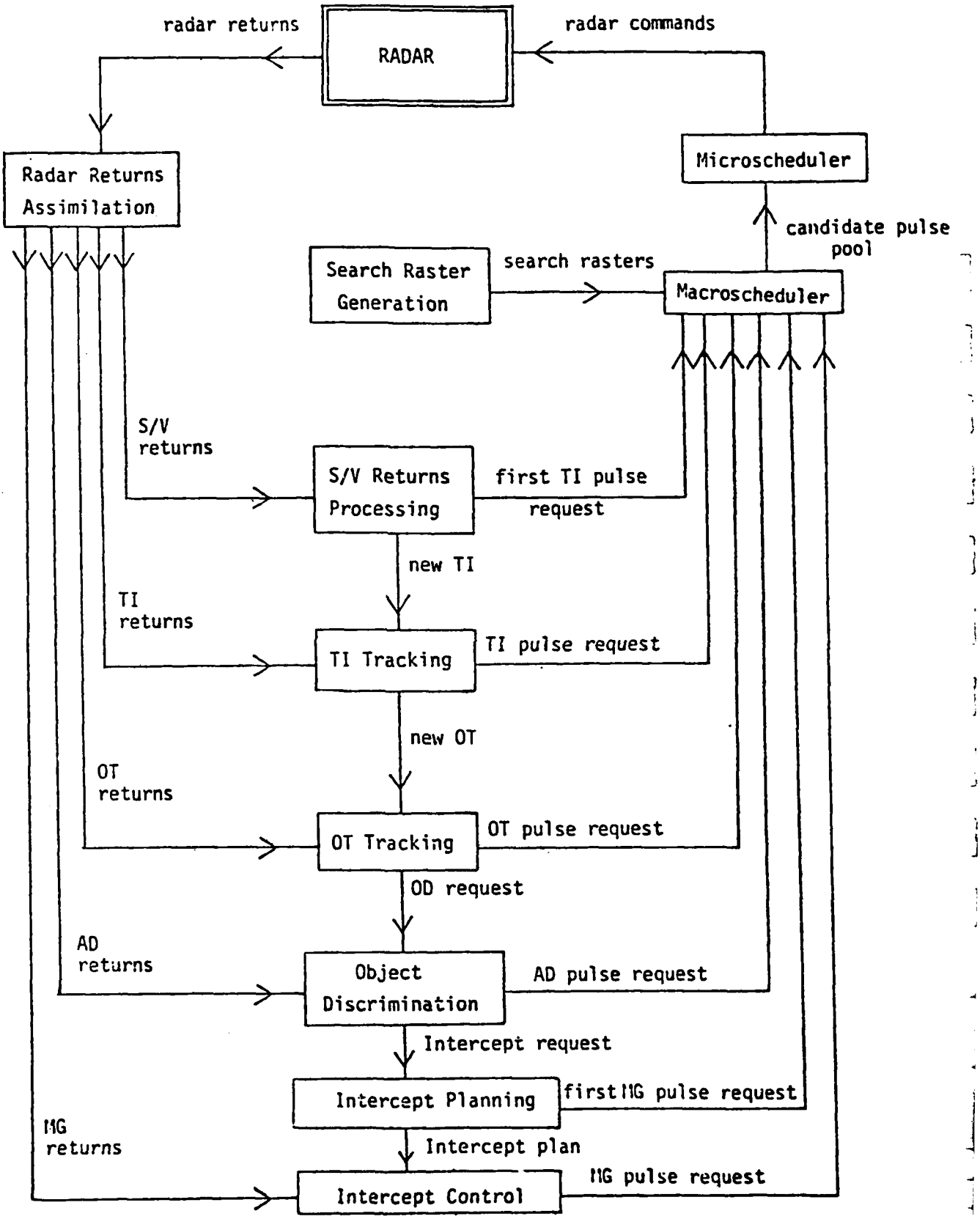


Figure ES-2. Process Overview

NETWORK SYNTHESIS AND EVALUATION

Two previously developed concepts for hardware architectures were evaluated - a representative Centralized Architecture and a Thread Architecture devised by McDonnel-Douglas. The evaluation procedure involving PAES is illustrated in Figure ES-3. The formal Process Definition provided the basis for defining the Process Description and Threat Scenario input segments for PAES. A third input segment was defined corresponding to each of the two Candidate Computer Architectures.

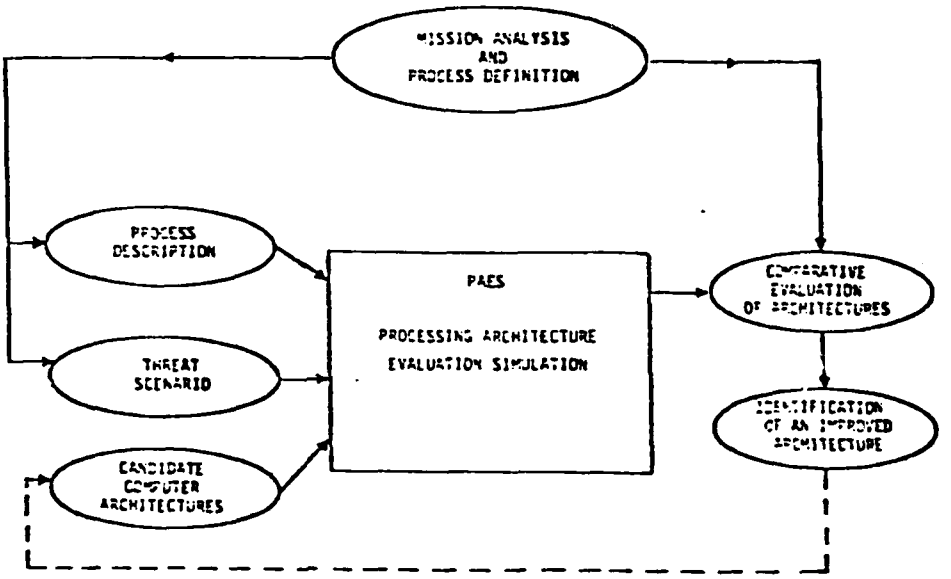


Figure ES-3. Network Synthesis and Analysis Procedure

Sample results of the evaluation of these two architectures are provided in Table ES-1. The results point out the increase in CPU requirement which is associated with distribution of the processing load to a set of computers dedicated to specific tasks, as is the case with the Thread Architecture. Further, one sees that the Thread Architecture requirements are more sensitive to the scenario.

The next step was to make use of the process analysis and the evaluation results for the Centralized and Thread Architectures to synthesize an optimal architecture. This process led us to identify an architecture of hybrid form.

Table ES-1. Evaluation Results for the Centralized and Thread Architectures

Architecture	CPU requirement (MIPS)*	
	Single-Spike** Scenario	Double-Spike*** Scenario
Centralized	47.67	44.94
Thread	62.17	54.01

* Application processing requirements plus allowance for operating system overhead

** 140 total objects introduced in a single wave over 100 ms

*** 140 total objects introduced in two waves, each of 100 ms duration and 1 second apart

THE HYBRID ARCHITECTURE

The recommended Hybrid Architecture is illustrated in Figure ES-4. The principal features of this architecture are:

Computing Nodes

- Special purpose processor for (1) Radar Return Assimilation, (2) control of the activity of the THREAD Processor, and (3) control of the RR (radar return) bus (RRA)
- Unit Resource Manager processor (URM)
- Special purpose processor for Track Memory access and update (MEM)
- Special purpose processor for Radar Pulse Scheduling and control of the PR (pulse request) bus (SCHEDULING)
- A set of processors of identical architecture (the "THREAD Processors") to perform thread processing (track initiation and update, discrimination, interceptor control).

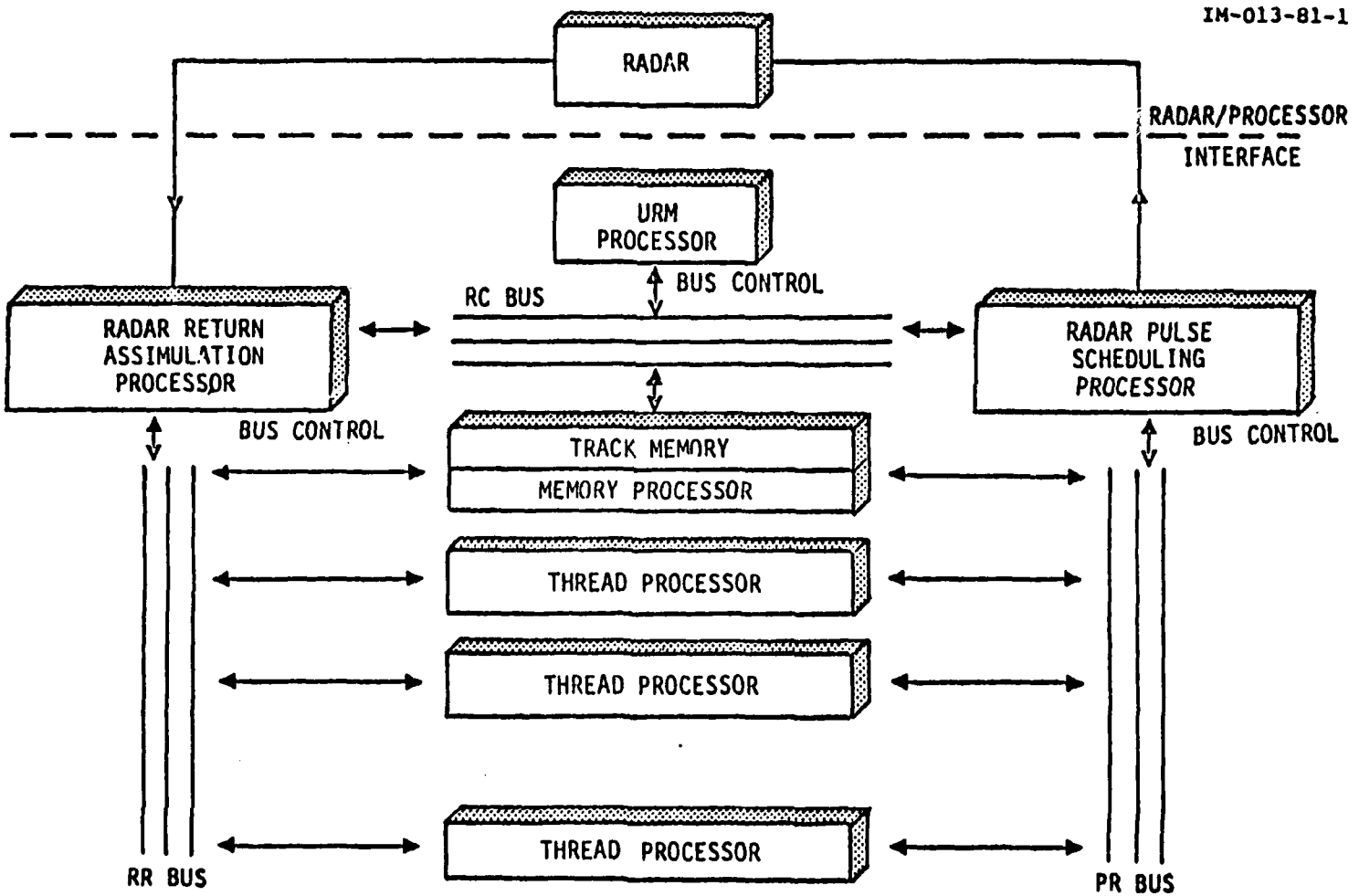


Figure ES-4. Recommended Architectural Approach for the BMD Site Defense Distributed Processor

The bulk of the processing is performed by the THREAD Processors. Analysis has shown that this partitioning into four dedicated processors and a multi-processor for various thread functions can in principal be realized without incurring the increase in total CPU requirement associated with the dedicated distributed computer organization of the Thread Architecture.

An analysis of port-to-port response requirements reveals that most of the thread processing can be accomplished with processors having .5 MIPS processing power. End game processing (interceptor control and associated object tracking) would require 1.25 MIPS processors. Thus, options for architectures of this generic form range from a "large" number of "small" processors, up to one or a few "large" processors. Further analysis, including costing, would resolve the choice of optimum size thread processors.

Communication Buses

- RR (Radar Return) Bus structured as a parallel 32b X 1 Megabit/sec channel under TDMA control, handling pulse returns and track records for use by THREAD processors

- PR (Pulse Request) Bus structured as a parallel 32b X 1 Megabit/sec channel under demand responsive control

- RC (Resource Control) bus.

This organization of the necessary data transfers takes advantage of the inherently uniform and predictable total loads, independent of scenario. These buses can easily be designed using current technologies.

Evaluation of this Hybrid Architecture yielded nearly identical results for CPU requirements as for the Centralized Architecture. Including estimates for the control of the THREAD Processors and for bus control, we obtained the estimates of CPU requirements provided in Table ES-2.

Table ES-2. Evaluation Results for the Hybrid Architecture

CPU Requirement (MIPS)	
Single-Spike Scenario	Double-Spike Scenario
37.63	35.97

The differences, as compared to the results for the Centralized and Thread Architectures given in Table ES-1, are due to: (1) the (potential) efficiency of the multi-processor structure of the THREAD processors in the Hybrid Architecture yields an application requirement nearly identical to that for the Centralized Architecture; (2) the clear functional partitioning in the Hybrid Architecture would entail substantially less operating system overhead.

CONCLUSIONS

The Hybrid Architecture advanced here offers a number of singularly attractive characteristics:

(1) The approach allows the shifting of the functions addressed by a computer during a scenario so that the system loading remains balanced, and no processing capability is idle during peak loading.

This characteristic depends on the centralized pulse-return scheduling function located in the RRA, to retain the simplicity of design, and the centralized database structure.

(2) Fault tolerance is embedded in the parallel design. THREAD processors are envisioned as performing constant on-line testing. When a fault is reported by a processor to the RRA, this THREAD Processor is immediately descheduled. The problem of providing high processor reliability is then concentrated on the special-purpose RRA, MEM, and SCHEDULING processors.

This characteristic depends on the centralized scheduling and the completely redundant structure of the proposed THREAD processors.

(3) The computing speed of the THREAD processors need not be over 0.5 MIPS, except for end game processing, where 1.25 MIPS processors are required. Thus, the actual capability selected for these processors would be based on a trade-analysis of cost, redundancy, and operating system complexity. Thus, there is an opportunity to minimize cost.

(4) Maximum loading of the processor segments is determined by the radar pulse rate limit. Thus, simple protocols for the BUSSES, and MEM, RRA, and SCHEDULING processors can be implemented.

The architecture requires a 1 Mwps at 32b/w BUS on the input and an identical BUS on the output. Given 10 accesses/pulse-return as a bound, 20,000 access/sec (50 μ-sec/access) are required to support a 2000 pps radar rate. These data and access rates are easily obtained with present technology.

(5) Both the RRA and SCHEDULING processors are special-purpose processors with control structures dedicatd to performing the required algorithms. These processors, and the MEM processor, would be designed with emphasis on fault tolerance. The RRA and SCHEDULING processors both are implemented as two-stage processors. The URM might also be a special purpose processor.

The architecture can be contrasted with the Thread Processing Architecture or the Centralized Architecture that were also examined in this study.

The Thread Architecture has a limited number of functions assigned to each computing NODE. This requires that redundancy be added to the NODE structure in order to achieve high reliability. The redundancy can be provided for a NODE either internally to each processor, or by providing multiple processors. Also, distinct hardware and software components may be required for different NODES, increasing the cost and complexity of design and development. A Thread Architecture has the characteristic that substantial application processing capability is not being used at the maximum loading point in the worst-case scenario.

The Centralized Architecture has the major fault of not taking advantage of the response-time/throughput loading disparity characteristic of the Site Defense Problem. The processing power must support the peak throughput loading. A very powerful and costly processor must be provided to meet this load. Managing the central processor requires a significant operating system overhead in addition.

VERAC has analyzed the loading and throughput requirements for the BMD Site Defense Processor. This analysis has motivated the development of an approach to the processing architecture that is characterized by parallel processors scheduled in real-time to perform needed functions. The approach also depends on a central, track database managed by a dedicated processor, and on a single scheduling and control point.

1.0 INTRODUCTION

The U.S. Army Ballistic Missile Defense (BMD) Systems Technology Program (STP) office supports a program to maintain and update technologies critical to the deployment of a BMD system. One part of that program, the Advanced Data Processing Subsystem Investigation (ADPSI), was directed toward the identification of key technical issues involved in the data processing subsystem, and the resolution of those technical issues which would be time-critical in a full scale development. As a part of the ADPSI effort, VERAC conducted this study of alternative network architectures for a distributed data processing (DDP) approach for the data processing subsystem.

In order to meet the technical objective of this study, VERAC extended a previously developed simulation tool for network flow analysis into a wholly new Processing Architecture Evaluation Methodology (PAEM), with a supporting tool, the Processing Architecture Evaluation Simulation (PAES). PAEM/PAES, described briefly in Section 2 and more fully in Appendix B, provides a highly disciplined and extremely flexible approach to characterization, design and analysis of (centralized and) distributed data processing systems.

In addressing the functions to be supported, existing software designs have been used, specifically those of TAP, the Tactical Applications Program [3,4]. These designs are tied to the use of the CDC 7700 computer; as a result, many aspects of these designs reflect CDC 7700-specific implementation considerations rather than the process supported by the implementation. A prime example of this is the partitioning of the software into tasks which collectively could be effectively schedule by TOS, the Tactical Operating System.

The third section contains a discussion of the effort to obtain a machine-independent description of the BMD functions. This effort was an essential first step in addressing DDP architecture design, and

further, yielded an extremely useful characterization for subsequent related analyses, and, ultimately, for design of the software on a DDP system. The effort we have conducted included the following steps: (1) an analysis of the existing BMD task designs, particularly those which were actually realized as code (radar interface, target and object tracking and discriminaton), to obtain a characterization of task code structure (functions, principal data strucure, and timing models); (2) an aggregation of tasks based upon associations necessary for principal functional requirements (e.g., tasks on a processing thread were generally put together); (3) a sequentially decomposition of these functions, ultimately to yield a set of PRIMITIVES, each of which is a smallest, logically cohesive unit. These PRIMITIVES served as the basic units allocated in development of a distributed processing architecture.

Performance requirements and design constraints have been obtained from available documents. The principal requirements which have a significant impact upon architecture are (1) the "port-to-port" response times associated with the processing of the various types of radar returns, and (2) the throughput or total computer processing load necessary to support all functions in selected scenarios. These requirements served to guide the Process Definition.

PAES was used, independently of a specification of computer network architecture, to perform an analysis of the loading associated with two BMD scenarios. This loading analysis provided valuable insights into the efficacy of candidate architectures and aided the synthesis of the Hybrid architecture.

Three types of architecture were examined: (1) centralized (roughly corresponding to the existing CDC 7700 implementation), (2) thread (closely corresponding to the MDAC paper design), and (3) hybrid, developed by VERAC as a part of this study. Evaluation of these architectures is given in Section 4, with further details contained in Appendix C.

Section 5 provides important conclusions of this study.

2.0 PAEM/PAES

This section briefly presents a methodology for supporting the analysis and design of the distributed data processing subsystems. This methodology played a critical role in this study in supporting the design and anslysis of alternative computer network architectures for the Terminal Defense data processing subsystem. In this application, the number and complexity of functions to be supported and the large number of hardware architectural options leads to a large, complex design effort. The purpose of the methodology presented here is to provide an efficient means of assessing HW/SW options at a level of detail sufficient for first cut designs, while, at the same time, realistically reflecting the dynamic processing load encountered in the Terminal Defense mission.

The Processing Architecture Evaluation Methodology (PAEM) presented here is a systematic procedure for modeling the system function required with hardware and software implementations. PAEM utilizes the Processing Architecture Evaluation Simulator (PAES), the tool which makes possible performance evaluation in a dynamic environment. Further detailed description of PAEM/PAES is provided in Appendix B.

2.1 Processing Architecture Evaluation

Developments in computer technology have introduced a new direction in the design of dedicated data processing subsystems. Where previously a single central computer would support requirements, the critical design effort was focused on software - processing algorithms and the application operating system. With the impending availability of processing elements with power in the 3-5 MIP range and therefore applicability of DDP for Terminal Defense missions, the architectural design associated with the network of distributed computers now is on a level with the software design.

The Processing Architecture Evaluation Methodology and the supporting simulation tool, PAES, supports two stages of the DDP HW/SW design process. The first stage is the Definition of the Process in

terms of the function to be supported, the environment (i.e., threat scenario) and the requirements. PAEM supports this by providing a formalized framework for representing the Process. Further, the simulation tool, PAES, provides an effective means of checking the consistency of the functional Process Definition by exercising the functional model in a realistic, dynamic environment. The flexibility of PAES is a particularly important asset at this stage, since a variety of process models and threat scenarios are easily introduced for analysis.

PAEM/PAES can provide invaluable support to the second stage of design in which functions are allocated to hardware elements and architectural characteristics of the DDP system are specified. PAEM provides a framework for specifying architectural elements, including gross computer characteristics, operating systems, bus protocols and data base management techniques, in a fashion that pinpoints critical design issues. PAES provides an evaluation of a specific HW/SW design as it would be expected to function in a realistic, dynamic environment.

PAEM/PAES fills an important gap in otherwise available analysis tools. Analytic tools (static, queueing network models) and static simulations, while, being flexible and efficient to use, cannot reflect the critical congestion effects which actually occur in the dynamic environment. The presently available tools which can provide useful measures of performance in the dynamic environment are complex simulations which are relatively inflexible - requiring recoding to examine substantial deviations in the process definition or in the DDP architecture - and expensive to run. These features severely limit the ability of these complex simulations to provide an effective HW/SW design aid. They are, however, valuable for final validation of performance of a design, because of their fidelity in performance prediction. PAES provides a dynamic simulation, yet retains flexibility in application by depending upon a table-driven simulator whose table elements constitute the definition of the process, scenario and architecture. Run time efficiency derives from the flow approximation at some cost in performance prediction accuracy. As a result, PAES

provides a means for rapidly assessing an alternative design, and can, in fact, be integrated as an important aid in the actual design process.

2.2 The Nature of PAEM/PAES

In the design environment where the computing resource is a single, large central computer, the hardware/software design process separates, after, a preliminary design step, into two largely independent, parallel paths. Integration occurs near the end of the development cycle. In contrast, when a distributed data processing resource is a candidate approach, the hardware/software design process is integrated throughout. So, while the use of distributed data processing technology can lead ultimately to substantial advantages, the associated design process is substantially more complex because it involves a hardware design effort and substantially more coordination between the software and hardware design.

The Processing Architecture Evaluation Methodology (PAEM), supported by the Processing Architecture Evaluation Simulation (PAES) is specifically designed to support an integrated HW/SW design effort for DDP systems.

The complexity inherent in integrated HW/SW designs is substantially relieved in PAEM by the separate representation of process (functional representation of software) and hardware architecture (functional representation of hardware and hardware-related effects). The first step in PAEM is an architecture-free Process Definition which focuses on representation of the process to be supported by the software, representation of the threat scenario and representation of the evolution of the process. The tool PAES supports this step, again, independently of any architectural effects, by providing for simulation of the process in a canonical dynamic environment which does not include the effects of overhead processing and delays associated with specific implementations. This capability allows for detailed checking of the Process Definition before any architectural effects are introduced.

PAES itself is structured into three functional components (illustrated in Figure 2.2-1):

(1) Input Specification - The process, architecture, scenario and output reports required are separately specified. Automated verification of structural consistency is provided by this component of PAES.

(2) Simulation - A deterministic, flow-oriented representation of the processing of "work-units" operates on tables defined by the Input Specification component. Process and architectural effects are separately simulated. This provides the basis for architecture-free process simulation. The overall modularity makes additions of new architectural effects modules relatively easy. This component generates files as directed by the Report Specification.

(3) Report Generation - A User-interactive component produces performance tables and plots from files generated by the Simulation component.

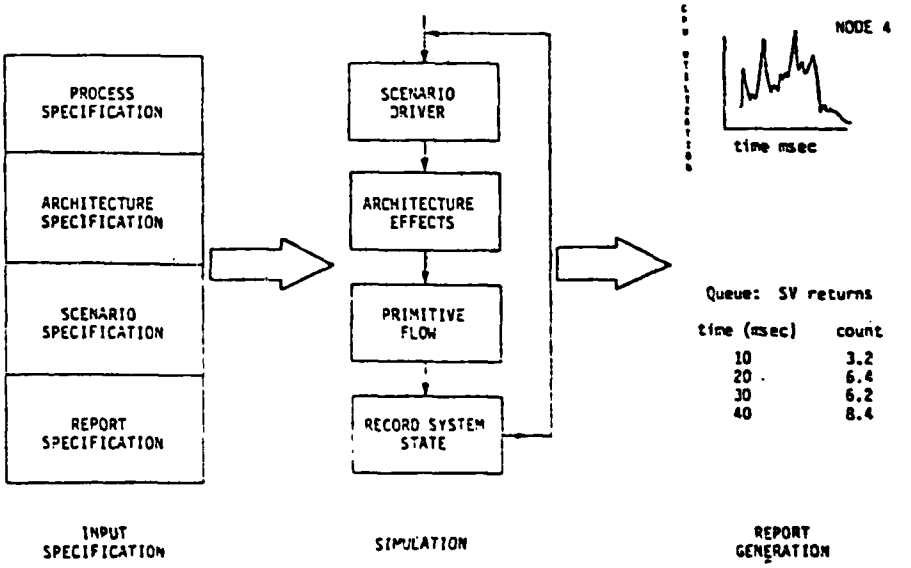


Figure 2.2-1 PAES - Processing Architecture Evaluation Simulation

2.3 Application of PAEM/PAES to Terminal Defense Data Processing Subsystem Analysis and Design

PAEM/PAES has an important role in two stages of the design process: (1) in the Process Definition/Functional Decomposition stage, and (2) in the analysis of architectural options. Each of these roles, as they relate to the Terminal Defense system, is described in separate subsections below.

2.3.1 Process Modeling

Process Modeling, illustrated in Figure 2.3-1, interfaces with a system level mission analysis and high level process characterization. The mission analysis produces a description of functions to be performed; the environment and, in particular, the threat scenario(s); and performance requirements. Functional decomposition is then undertaken to elaborate the description of functions into a set of subfunctions each of which is a logical, "primitive" process element. It is these primitives which are represented in PAEM/PAES.

In PAEM/PAES each processing primitive is represented in a standard format. This is illustrated by the representation for the primitive ANGRP in Figure 2.3-2. The principal elements of the representation are: (1) input queue buffer, (2) processing load representation in terms of (estimated) machine language instructions executed for each input "work-unit", (3) one or more accesses to a resource which may reside inside or outside of the computing node supporting the processing represented by this primitive, and (4) an output queue buffer. This is one of the simpler processing primitives. Other primitives involve several inputs, several external accesses (e.g. to data bases) and/or several outputs - but all primitives involve a single processing load model.

The high level process characterization is used to develop a process evolution model. For Terminal Defense this model reflects the sequence of processes which are triggered by the initial detection of an RV, ghost, decoy, fragment, or other "object".

The Processing Architecture Evaluation Methodology embodies a formalized approach to Process Definition, including a standard format for representation of subfunctions, scenario inputs and process evolution model. Use of PAES assists the Process Definition by providing a tool for checking out the mutual consistency of these elements of the process in a realistic, dynamic environment.

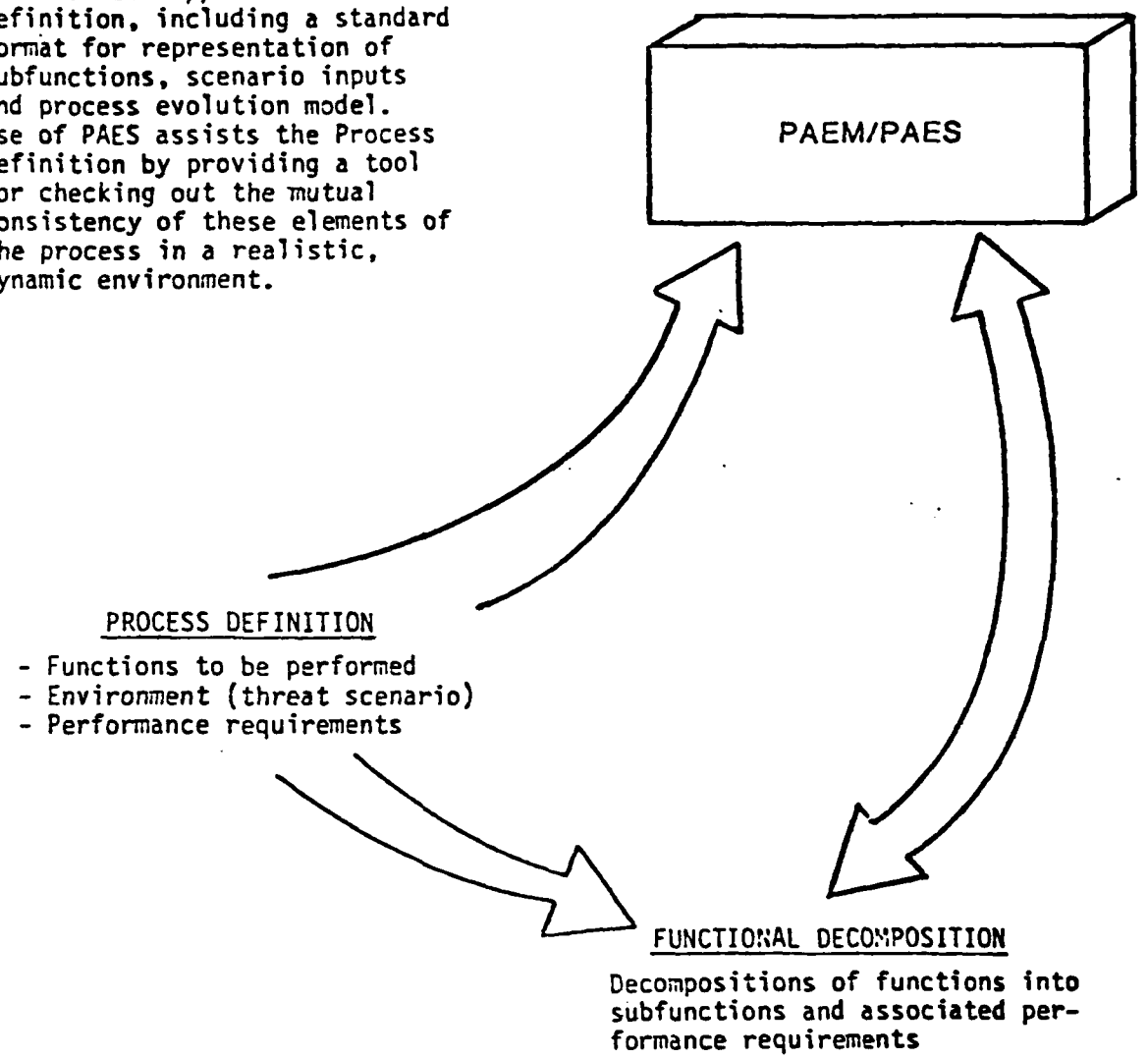


Figure 2.3-1 Role of PAEM/PAES In Process Definition

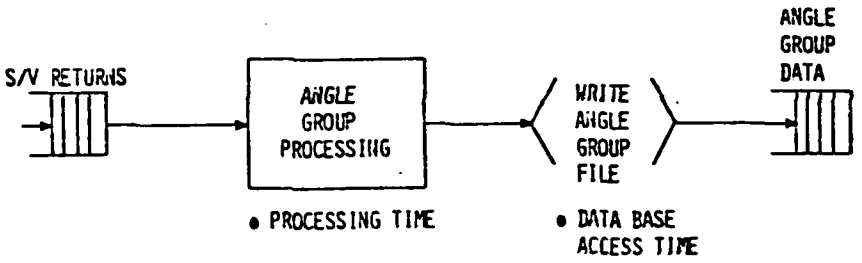


Figure 2.3-2 Characterization of a PRIMITIVE

PAES assists in this Process Modeling stage by providing a mechanism for validating the model in a realistic, dynamic environment.

The Process Evolution Model is determined by combination of the description of the threat and the high level description of the engagement process. For example, these factors indicate the average time spent in each stage of processing. These times, together with a specification of the mix of object types, determine the splitting parameters.

The tool PAES provides valuable support to Functional Decomposition and Process Evolution Modeling by exercising the explicit, parameterized representation of processing primitives, with the process model embedded, in a canonical dynamic environment. The Report Generation capability provides detailed views of internal processing (e.g., queue counts and flows) to assist in this validation effort.

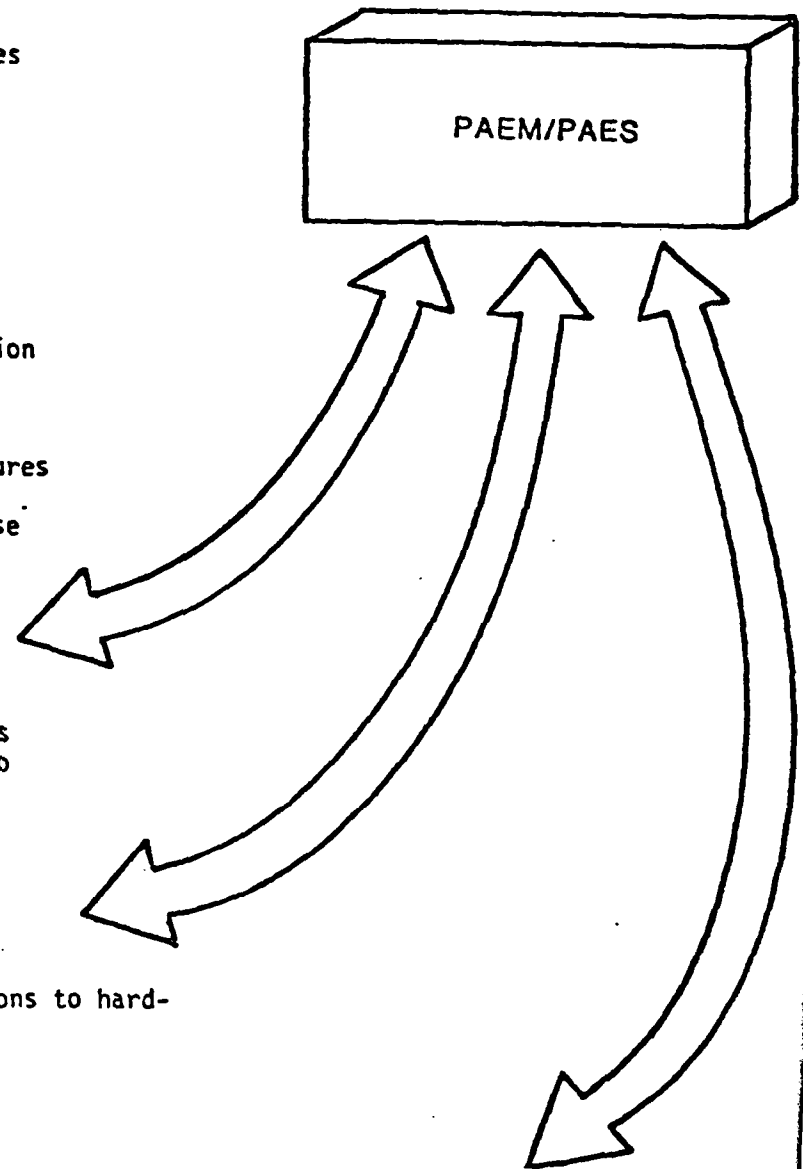
An extermely important feature of PAEM in this Process Modeling stage is the explicit representation provided which can serve as the context for discussion amongst the various contributors to an analysis and design effort. Assumptions about critical aspects of the process algorithms, in terms of processing loads, data files access requirements and interrelationships as defined by the process evolution model, are made explicit so that they can be the object of technical criticism, refinement, and, ultimately, consensus.

2.3.2 Evaluation of Architectural Options

The second and equally significant role for PAEM/PAES in DDP HW/SW analysis and design is in the evaluation of architectural options, as illustrated in Figure 2.3-3.

The Processing Architecture Evaluation Methodology supports successive HW/SW design stages through explicit representation of the allocation of subfunctions to computing Nodes (i.e. processors), the architectural characteristics of these Nodes, and the communication Buses and Resources associated with the Node set. The supporting PAES

The Processing Architecture Evaluation Methodology supports successive HW/SW design and evaluation stages, through explicit representation of the allocation of subfunctions to computing Nodes and of the architectural effects of these Nodes, and of communication Buses and Resources. The supporting PAES program provides for distinct, data-specified representation of functions, architectures and scenarios in order to provide maximum flexibility in efficiently examining HW/SW alternatives. The simulation and associated report generator provides a thorough view of performance in a realistic, dynamic environment, providing such measures as CPU utilization, buffer queue dynamics and port-to-port response times.



SOFTWARE FUNCTIONAL SPECIFICATION

Characterization of process algorithms corresponding to subfunctions

ALLOCATION

Mapping of subfunctions to hardware architecture

HARDWARE FUNCTIONAL SPECIFICATION

Characterization of computers, peripherals and communication buses

Figure 2.3-3. Use of PAES in Architecture Evaluation

program provides for distinct, data-specified representation of functions, architectures and scenarios in order to provide maximum flexibility in efficiently examining HW/SW alternatives. The simulation and associated report generation provides a thorough view of a performance in a realistic, dynamic environment, providing such measures as cpu utilization, buffer queue dynamics and port-to-port response times.

The design and analysis procedure is illustrated in Figure 2.3-4. The Process Modeling, accomplished as the first step of design/analysis, serves to define the process and threat scenario. Computing network structures are postulated and primitives are allocated to computing nodes, bus organization and protocols are defined, and external accesses are resolved to define candidate architectures.

The program PAES is applied to each candidate architecture to produce performance measures for purposes of comparison to requirements and to results for other candidate architectures. Critical measures are port-to-port response time on critical threads and cpu utilization.

The flexibility in the use of PAES, particularly in specifying architectural alternatives, greatly assists in the isloation of critical functions that require special-purpose architectures or algorithm modification in order to achieve balanced cost/risk implementation.

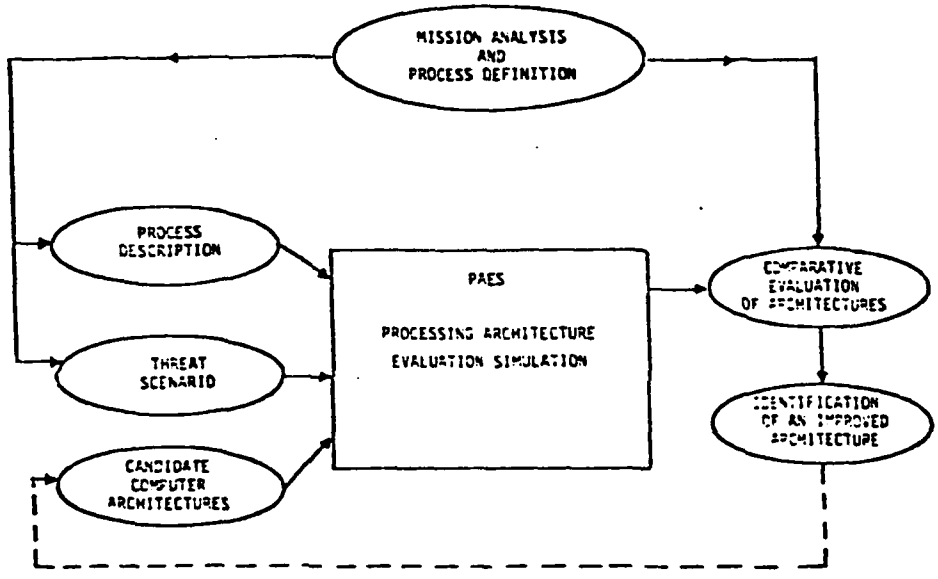


Figure 2.3-4. Network Synthesis and Analysis Procedure

3.0 PROCESS DESCRIPTION AND ANALYSIS

The purpose of this section is to present an overview of the process description developed for the BMD terminal defense data processing function. Furthermore, implications for architecture are derived by application of PAEM/PAES to this process description.

3.1 Functional Overview

The functions to be supported by the data processing subsystem in the Terminal Defense System (Underlay) are illustrated in Figure 3.1-1 (taken from [1]). At the highest level, these functions are partitioned into three groups:

(1) defense unit or "Non-Module Command DU", indicated in the leftmost box,

(2) C^3 interface, and

(3) *module functions*, indicated in the rightmost box headed by "Module Command DU".

The top level unit functions that have been identified are prescribed in the following:[1]

Unit Resource Management (URM)

The URM function maintains the object status information for each object in track. It accepts and processes messages from the ODD function to establish track on objects, and from the OT/OD functions to maintain status on objects. It provides processing to perform allocation/reallocation of DP and radar timeline resources. It provides to the OT, OT, ODD, and USC functions the DP and radar timeline

[1] These descriptions are adapted from [2]. References to KMR (Kwajalein Missile Range) test functions have been deleted.

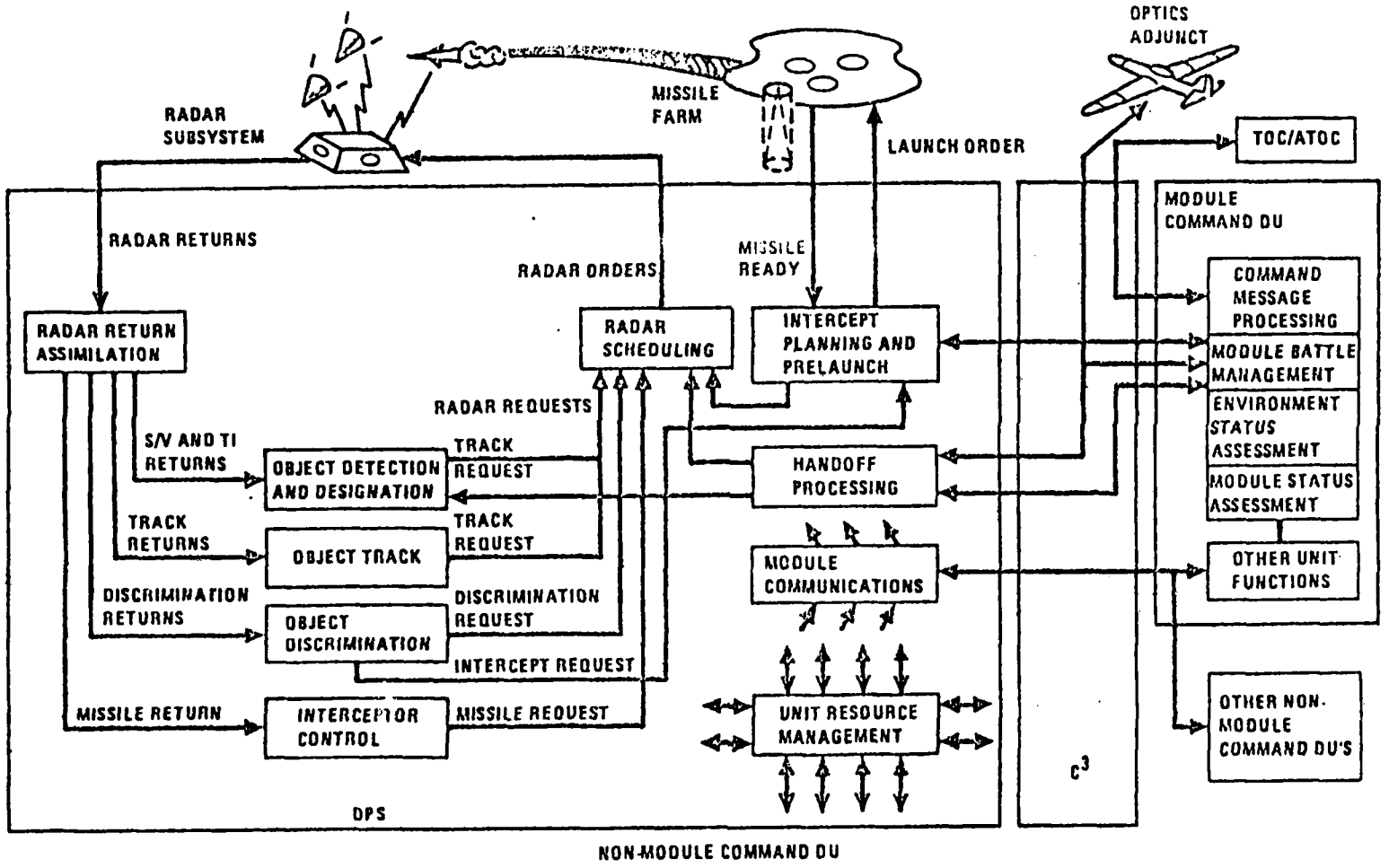


Figure 3.1-1. Unit and Module Functions in the Terminal Defense System (Underlay). Taken from [1].

resources available for data augmentation and maintains data received from these functions on the level of resource utilization.

Radar Return Assimilation (RRA)

This function processes all return messages from the Systems Technology Radar (STR). It evaluates the validity of each message from the STR, and sorts messages in order to determine subsequent Engagement Software (ESW) functions for processing.

Radar Scheduler (RS)

This function provides the TAP interface with the STR for scheduling and transmitting radar command messages. The RS function provides the STR with all the data required to transmit pulses and associate and process returns to the ESW. The RS function receives radar pulse requests from other functions, and schedules radar pulses as limited by radar timing and energy constraints imposed by the Unit Resource Manager (URM).

Unit Search Control (USC)

This function (not illustrated in Figure 3.1-1) stores and provides the RS function information needed to initiate scheduling of identified search areas for reentry vehicles. As such it identifies the set of prestored search beam requests (including the designation of normal search versus data augmentation search), their execution sequence, and associated pulse rates to be scheduled by the RS function.

Object Detection and Designation (ODD)

This function processes all search/verify target reports in order to prepare state vector estimates for the initiation of tracking and to reduce the non-threatening and redundant target load in track. It first associates valid detections in the search/verify report into angle groups. For each angle group a track initiate pulse sequence is determined.

The ODD function eventually designates each target (valid detection in the S/V report) as a ghost, object or non-threatening object (as determined by a non-threatening velocity vector). This designation is based on track initiate radar return information for each angle group. Information on designated objects are passed on to the OT function for track processing.

Object Track (OT)

The OT function provides the capability to establish and maintain track on objects which pass the track initiation process. It evaluates quality of tracking returns, updates object state and error covariance from valid returns, and determines the beam pointing parameters for additional tracking pulses. The OT function monitors status on each object and writes status change data to URM.

Object Discrimination (OD)

The OD function provides the capability of classifying an object as an RV, tank, decoy or fragments. If an object is classified as an RV, then OD will determine the class of RV and where the impact point is expected. An option is provided to drop objects from track via a notification to OT for those objects determined to be fragments. In order to perform this processing, OD will request the scheduling of active discrimination radar pulses by the RS function.

The interface to the Command, Control and Communications (C^3) Subsystem is performed by the following function (indicated as Module Communications in Figure 3.1-1):

Defense Module Communications (DMC)

This function handles interfaces between the ESW and the Command, Control and Communications Subsystem. As such it processes all high and low speed communications traffic and performs message error processing.

3.2 Functions Represented by the Process Description

For purposes of detailed analysis, a portion of the set of functions has been selected. These functions are illustrated in figure 3.2-1. The functions included here are:

- radar returns assimilation
- radar scheduling (including search raster generation)
- S/V returns processing
- angle group tracking (Track Initiation)
- object tracking
- object discrimination
- intercept planning
- interceptor control.

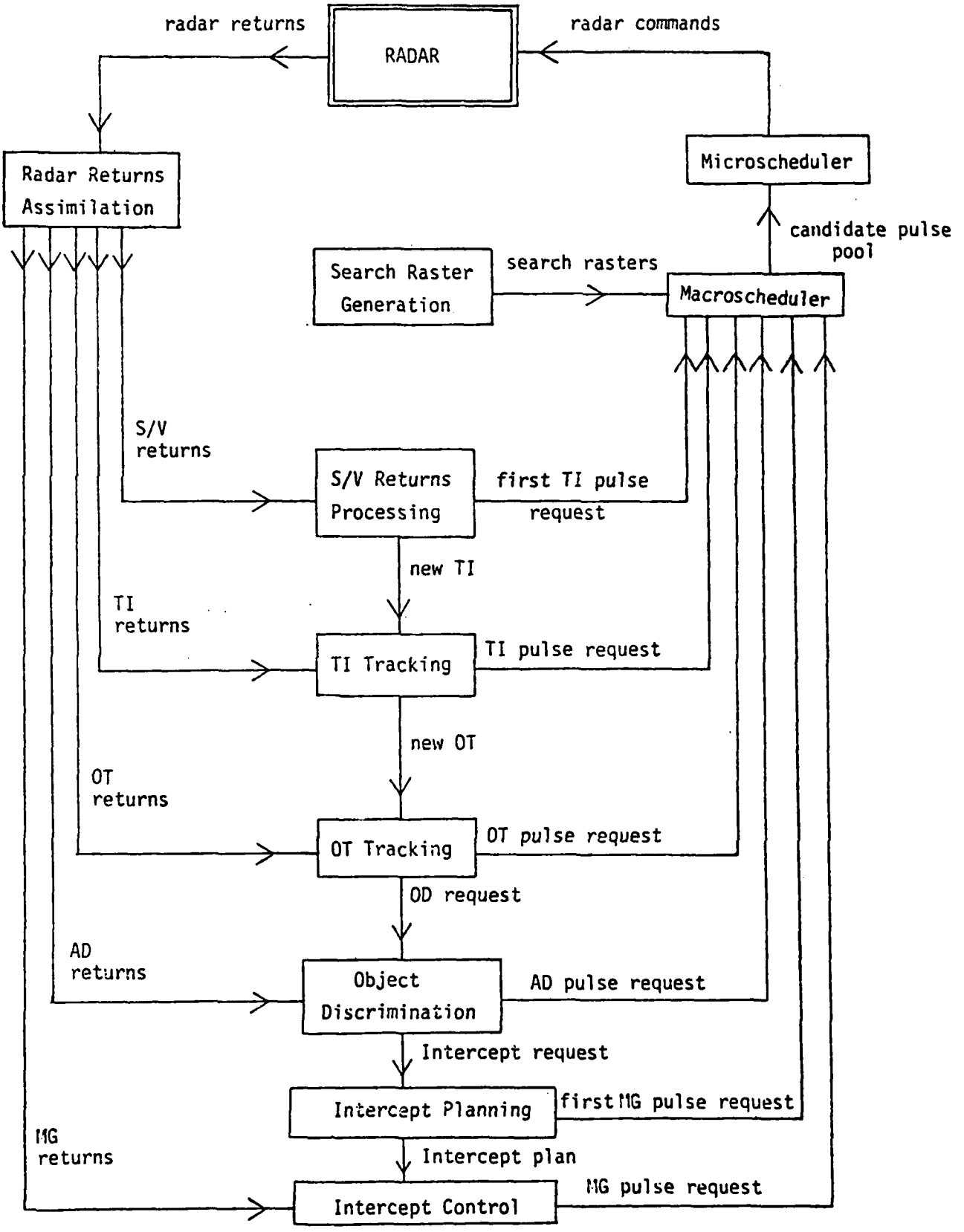
Not included are:

- unit resource management
- communications interface
- model level functions.

This selection was based upon the fact that the selected functions constitute nearly all the processing load and include all critical threads for response time considerations. The remaining functions are not well-described in existing documentation.

3.3 Functional Decomposition

The development of the detailed process description was based upon a critical review of available software documentation ([1], [2]). The processing of TASKs was examined to identify structure, timing models and required data accesses and transfers.

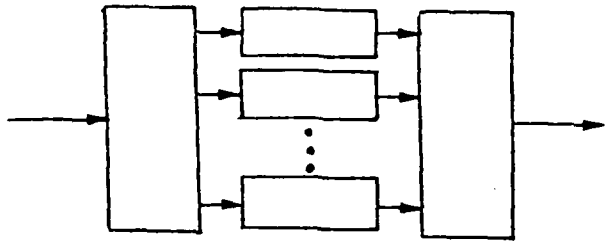


3.2-1. Process Overview

Structure was identified in terms of three general elementary structures: parallel, sequential-repeated and sequential-distinct, as described further here.

Parallel

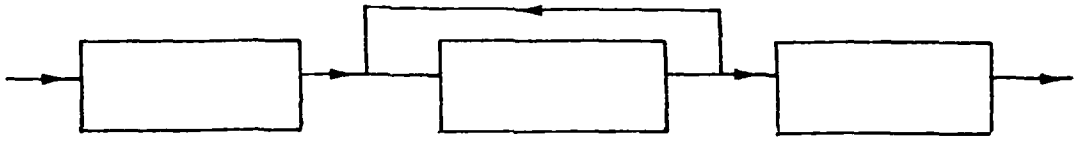
The function substantially consists of alternate, and logically distinct, processing paths, depending upon the specific nature of the input instance. Pictorially, the functional structure is of the form:



where the initial and/or final stage of processing may be absent.

Sequential-Repeated

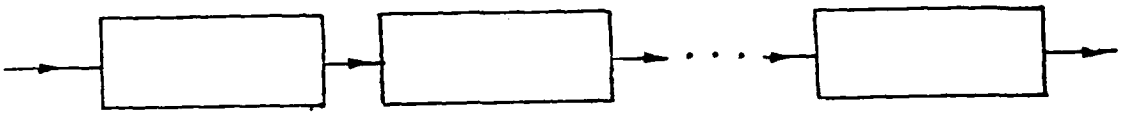
The function substantially consists of a sequence of processes, a large portion of which are identical in nature, as for instance in an iterated loop. Pictorially, the functional structure is of the form:



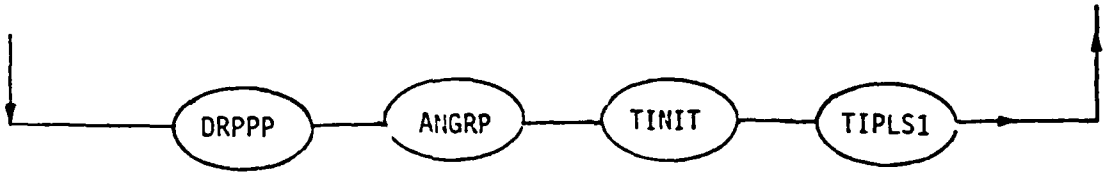
where the initial and/or final stage of processing may be absent.

Sequential-Distinct

The function substantially consists of a sequence of processes, logically bound and distinct in nature. Pictorially, the functional structure is of the form:



The analysis was recorded in the form of a set of one or more PRIMITIVEs corresponding to each TASK. For example, the S/V returns processing function, implemented in the TASK TDRP, in represented as:



Each named ellipse is a PRIMITIVE function. This diagram illustrates the sequence of processing events.

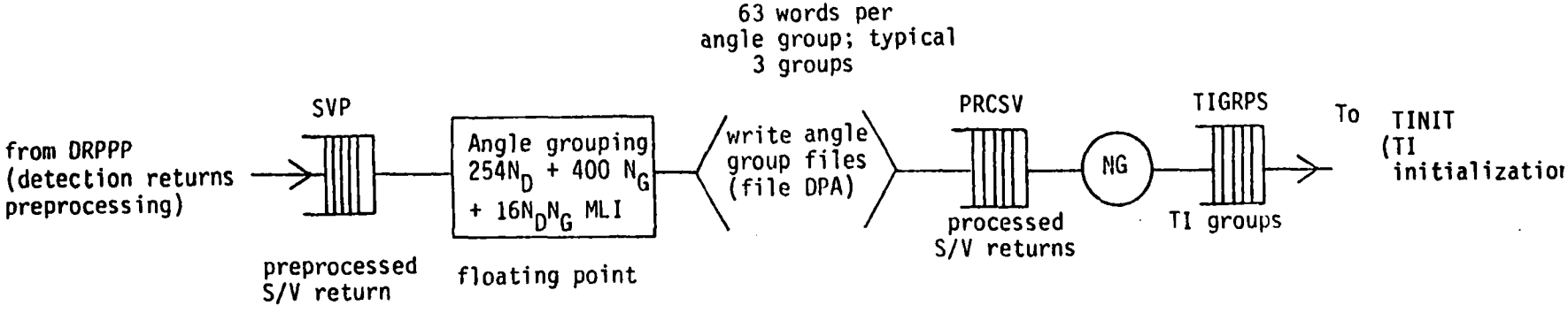
Each PRIMITIVE details the inputs, outputs, processing load (CDC MLI instructions), significant data accesses and transfers, and scenario-dependent evolution of processing. For example, the PRIMITIVE ANGRP is illustrated in figure 3.3-1.

In Appendix A, a complete, detailed process description is provided for the function of figure 3.2-1. One particular feature of this description merits attention here. We have adopted a flow-oriented

Primitive: ANGRP

Angle group processing

Detection returns processing function



Nominal parameters

N_D: 10 (5-15)

N_G: 3 (2-4)

Implies 4220 MLI

Figure 3.3-1. PRIMITIVE ANGRP, illustrating two mechanisms for representing data access and transfer

representation of the evolution of the status of objects (and interceptors) as they are detected, tracked, classified and intercepted. These parameters appear in the final splitting to obtain outputs in each PRIMITIVE.

In Figure 3.3-2, the complete set of primitives are indicated. The flow indicated corresponds to the sequence of processes corresponding to pulse returns. The general layout corresponds to the process overview previously presented in Figure 3.2-1. In this figure one can see this detail and corresponding processing of pulses associated with previously identified process evolution states. In Appendix A, details of the process evolution model are provided.

3.4 Process Evolution

To provide a representation of the process evolution we have adopted a finite-state model. The process states are indicated in Table 3.4-1. Objects begin in the state "SV", i.e., have associated search/verify returns, and evolve through states TI → OT1 → OT2 → OT3 → OT4 → OT5→ complete. Some objects are dropped at an intermediate stage. Further, entering OT3 is associated with the generation of active discrimination pulses, and OT4, with tracking while planning intercept, and OT5, with tracking while intercepting.

The overview of the process evolution is illustrated in Figure 3.4-1.

The specific scenarios assumed in evaluating alternative network architectures is characterized by the following mix of objects:

Redundants	8
Ghosts	10
Fragments	20
Decoys	70
RV's	32
Total Objects	140

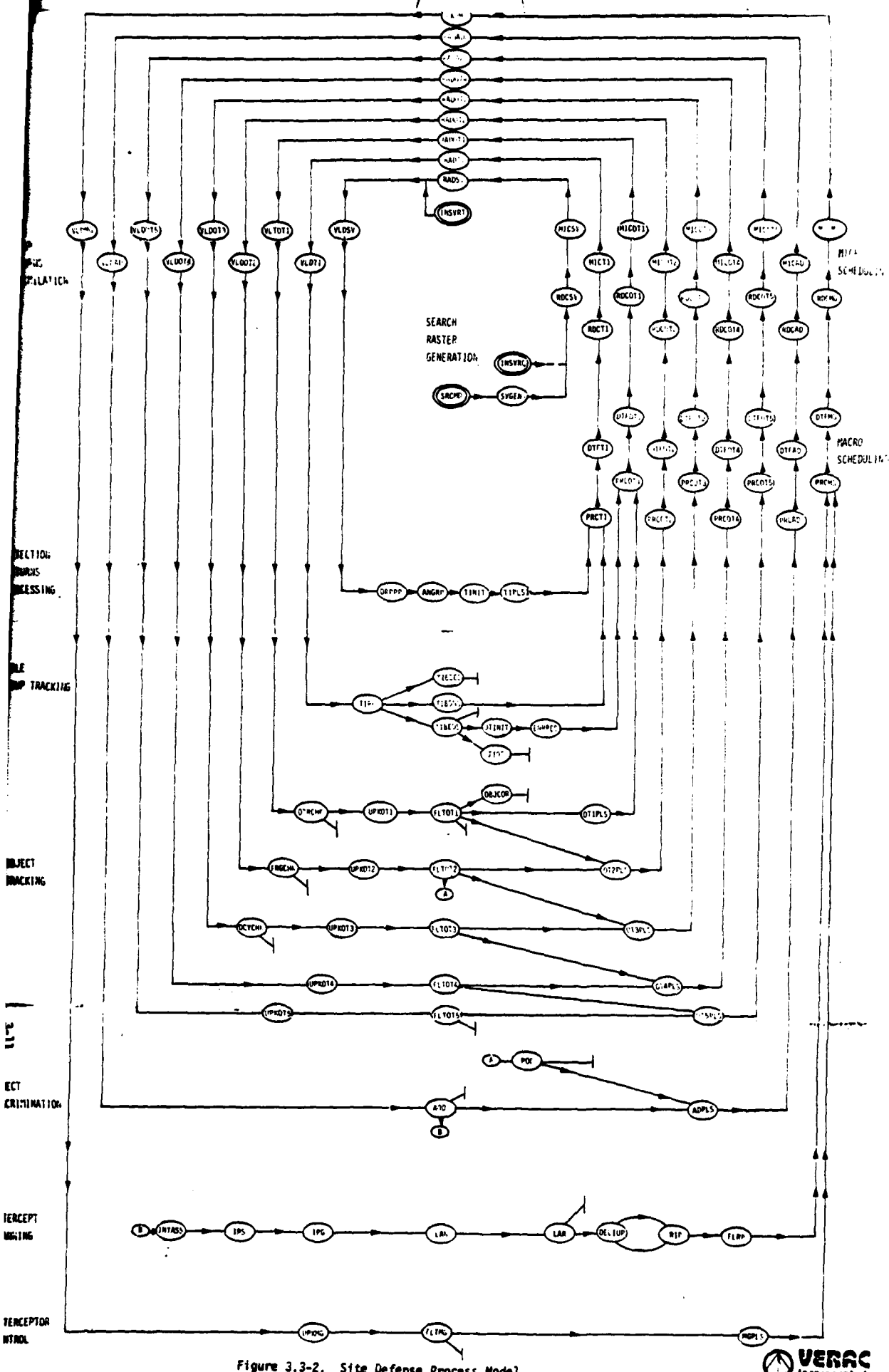


Figure 3.3-2. Site Defense Process Model

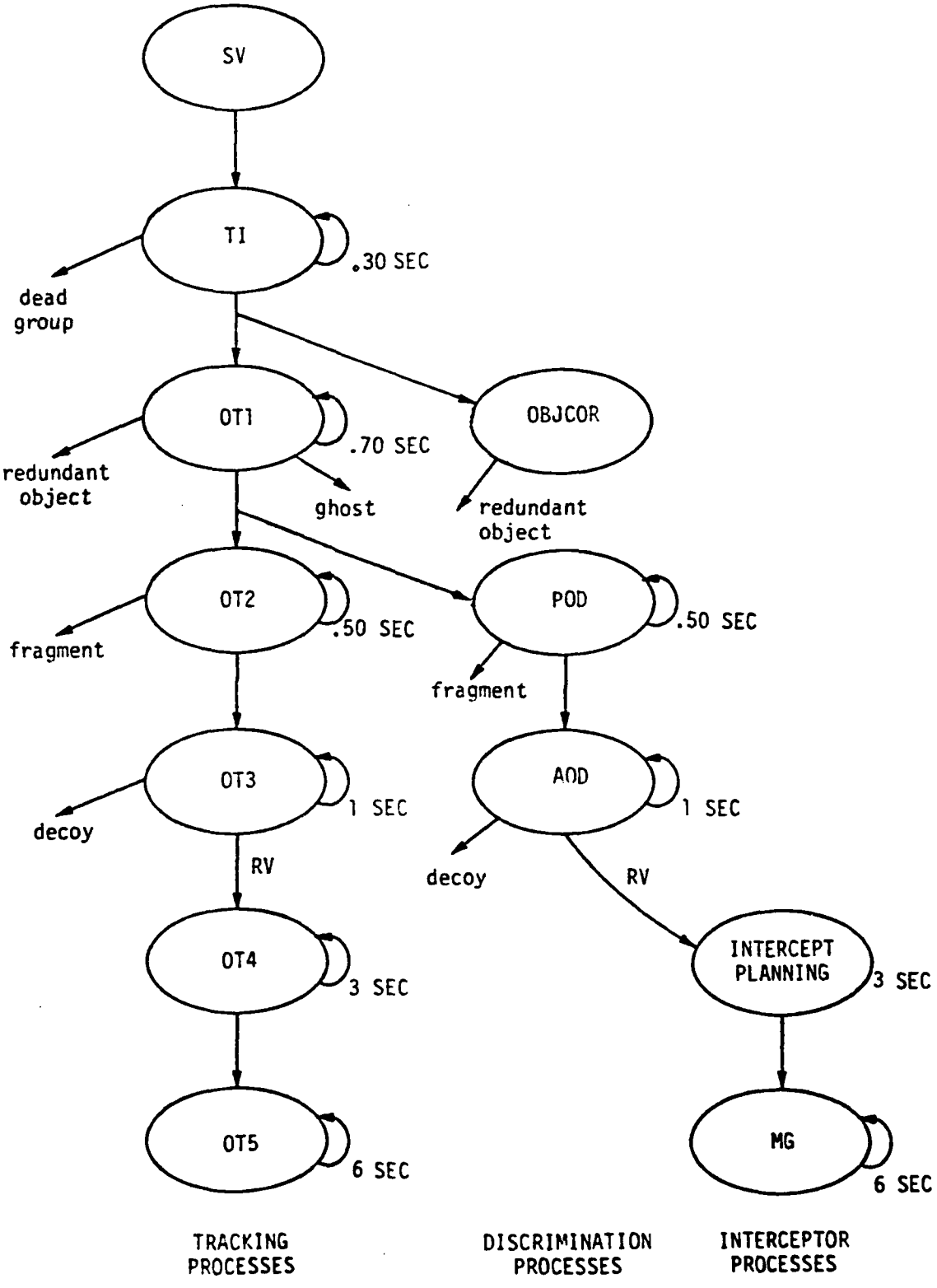


Figure 3.4-1. Process Evolution

Table 3.4-1 Process Evolution States

SV	search verify processing
TI	angle group tracking (track initiation)
OT1	object tracking before any object discrimination
OT2	object tracking with passive object discrimination
OT3	object tracking with active object discrimination
OT4	object tracking during intercept planning
OT5	object tracking during intercept
POD	passive object discrimination for unclassified objects (objects in state OT2)
AD	active object discrimination for unclassified objects (objects in state OT3)
MG	missile guidance

These objects were generated by inserting an S/V returns time history into the S/V radar return path (see Figure 3.3-2) through the use of the PRIMITIVE INSVRT. Two scenarios were examined by variation of the S/V returns time history (and with the object mix fixed as previously indicated). The S/V returns time history for each are illustrated in Figure 3.4-2. The first of these, the "single-spike" scenario involves introduction of 75 S/V returns in the first 100 msec. The second, the "double-spike" scenario, introduces 40 S/V returns in the first 100 msec and 35 S/V returns in the interval from 1200 to 1300 msec.

The single-spike scenario presents a "worst-case" for loading. In the absense of queueing, this scenario produces the maximum peak loading for any portion of the process amongst all scenarios with the same total number of S/V returns distributed over more than 100 msec. In particular, NODE loading has a maximum peak for this scenario.

The double-spike scenario represents a more typical situation and in fact closely approximates a scenario which has been examined by McDonnel Douglas in their simulation analyses.

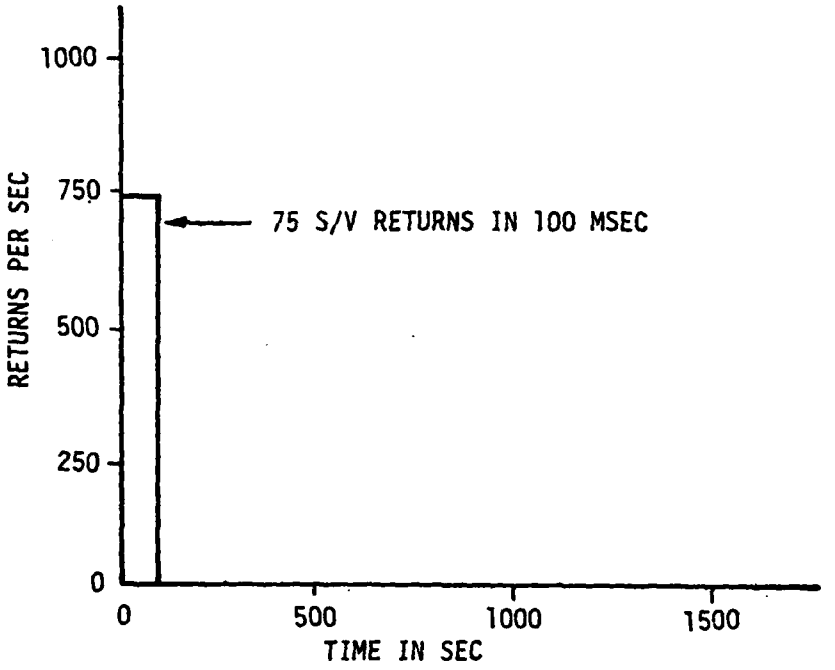
3.5 Response Time Requirements

The performance requirements for the site defense processor to be used in this study were provided by McDonnel Douglas Corporation. These are response time requirements along certain processing threads. Only unit level threads are of concern here. Table 3.5-1 presents the Port-to-Port thread definitions from radar subsystem to radar subsystem as defined by McDonnel Douglas in terms of TAP tasks.

Also indicated in this table are the threads which were modeled in PAES (any thread can be modelled), and the corresponding nominal response time requirement.

In using PAES, the set of PRIMITIVES of the form ".DTF" were used to insert delays which yielded the required response times. This device was important in maintaining the desired temporal evolution of the process.

(a) Single-Spike Scenario



(b) Double-Spike Scenario

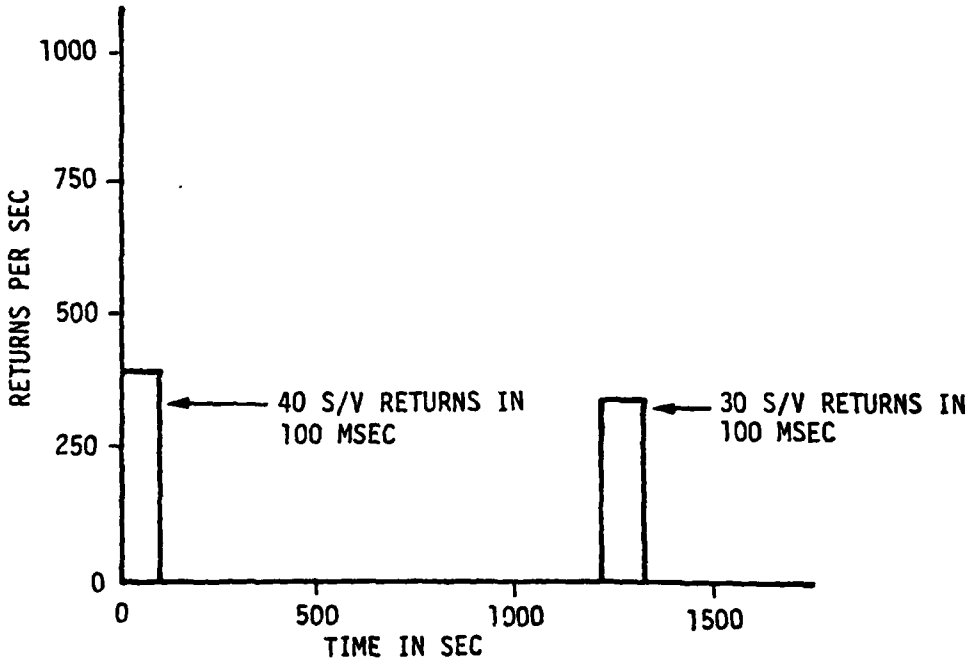


Figure 3.4-2. S/V Returns Time Histories For Two Scenarios

Table 3.5-1. Port-to-Port Thread Definitions

Thread	TAP Tasks	Measured by PAES	Nominal Requirement (msec)
Verify Return	TRIP-TDRP-TRIP	✓	100
Track Initiate	TRIP-TIRP-TSBT-TRIP	✓	50
Last TI	TRIP-TIRP-TSBT-TFII-TREQ-TRIP	✓	50
Normal and Post-Commit Track/Maintenance Track	TRIP-TOTT-TRIP	✓(✓)*	50 (25)
Drop Track/Track Rate Change	TRIP-TOTT-TREQ-TRIP		
Passive Discrimination/ Track to Discrimination Turnaround	TRIP-TOTT-TPOD-TRIP		
Active Discrimination Request/Drop Track	TRIP-TOTT-TPOD-TREQ-TRIP		
Active Discrimination	TRIP-TAOD-TPOD-TRIP	✓	25
Interceptor Guidance and Track	TRIP-TICT-TRIP	✓	20
Pulse Replacement/ Reschedule	TRIP-TRIP		
Object Reacquisition	TRIP-THAP-TRIP		

*Endgame tracking has the move stringent requirement of 25 msec.

3.6 Use of PAES in Process Definition and Analysis

The simulation program PAES was used extensively in the preliminary stages before architecture evaluation in arriving at a correct process definition and in providing an evaluation of the inherent loading requirements independent of any particular hardware architecture. In the two subsections which follow, we describe the use of PAES in each of these two areas.

3.6.1 Verification of Process Model

The process model is enbedded into the primitives in two fundamental ways. The structure of the evolution and the relative numbers of objects evolving at specific points in the process are represented in the branching at the output of corresponding PRIMITIVES. Simple analyses, presented in Appendix A, were used to deduce branching or splitting parameters which would yield the correct mix of objects.

Secondly, the set of PRIMITIVES of the form ".DTF" were used to introduce delays along each radar-radar thread to yield the desired response time, as inidcated in Table 3.5-1.

PAES was exercised on increasingly larger segments of the process to verify (and indicate needed corrections to) the process model. Final confirmation is illustrated by (1) the accumulation of number of objects at successive stages of the process, Figure 3.6-1; and (2) the thread port-to-port response times displayed in Figures 3.6-2 through 3.6-7. The latter figures show close agreement between modeled response times and requirements. All figures here relate to the double-spike scenario. Similar confirmation was obtained for the single-spike scenario.

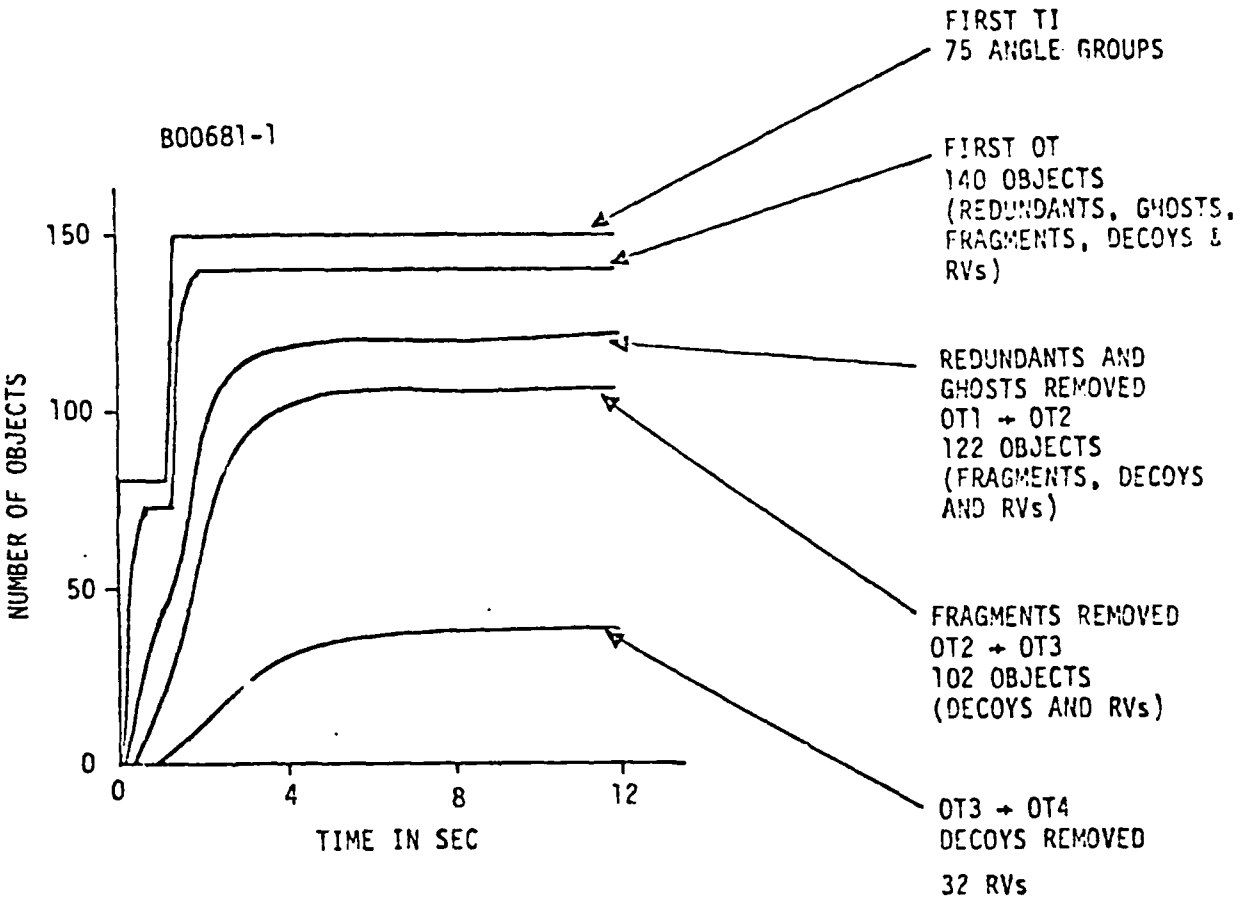


Figure 3.6-1. Accumulated Number of Objects Processed at Successive Stages

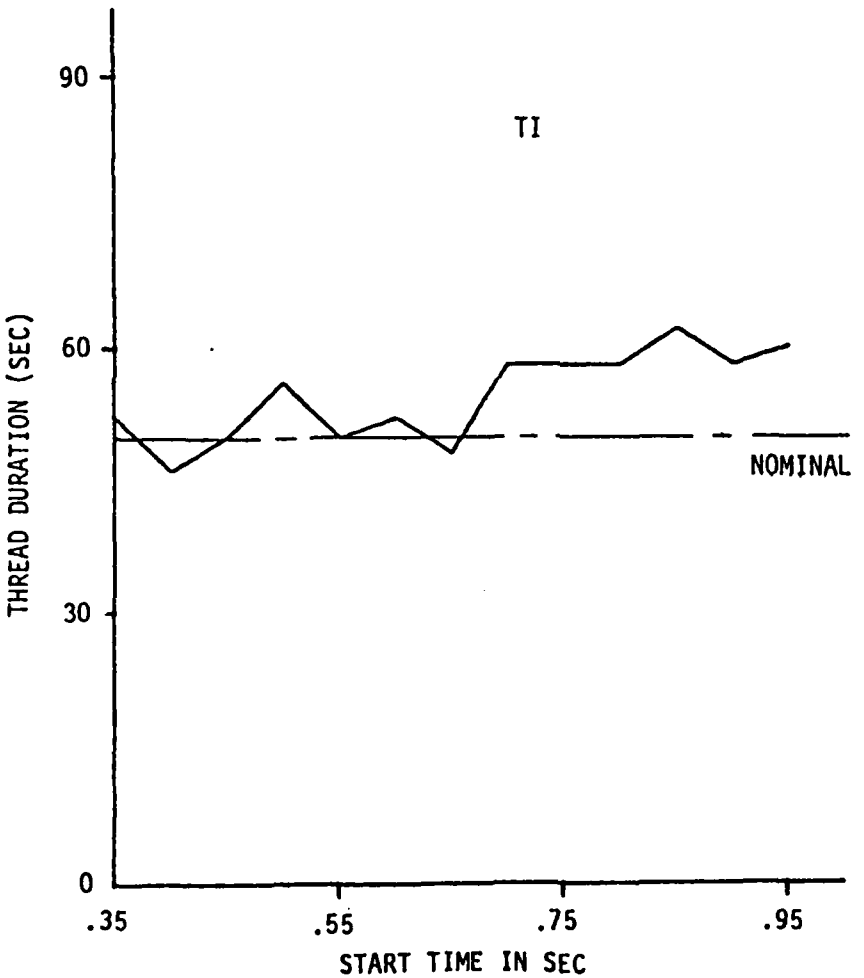


Figure 3.6-2. Track Initiate Thread Port-to-Port Response Times

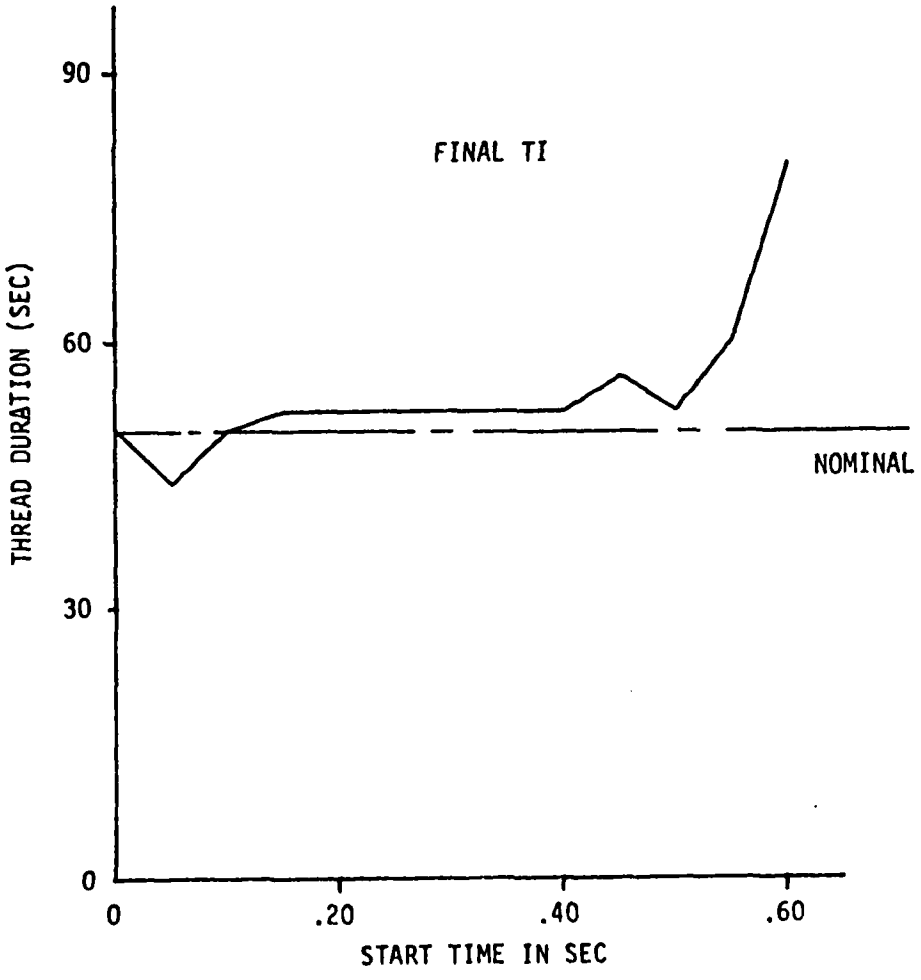


Figure 3.6-3. Last TI Thread Port-to-Port Response Time

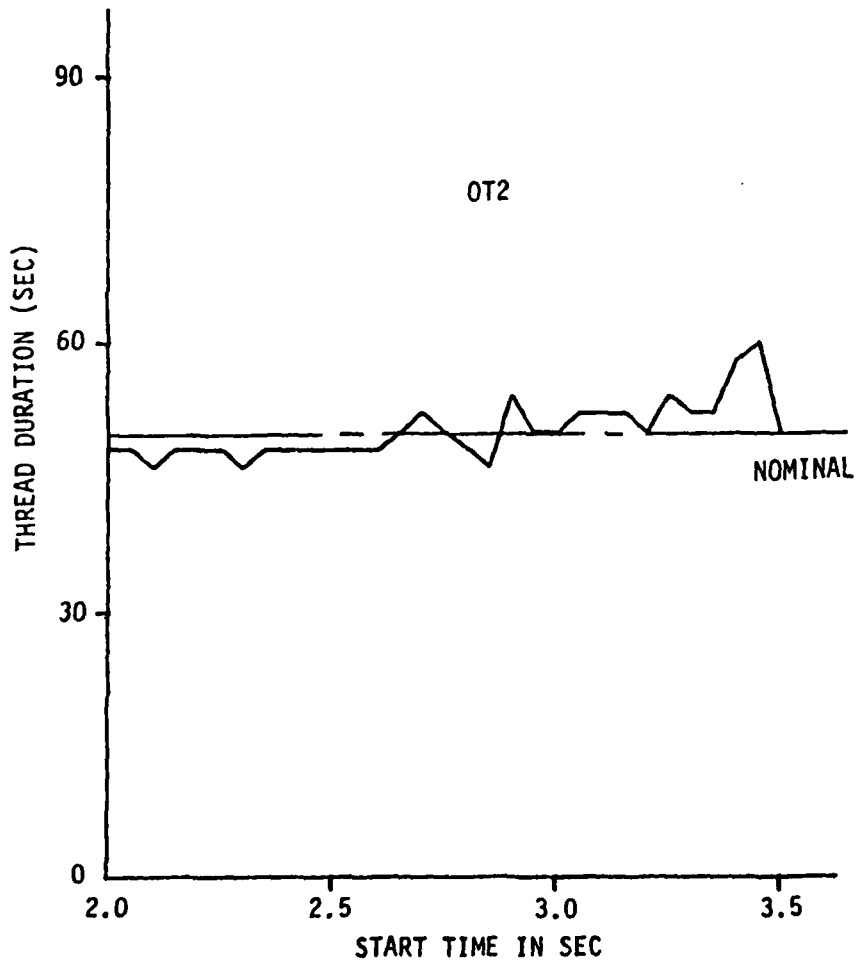


Figure 3.6-4. Normal Track Thread During Passive Discrimination Port-to-Port Response Time

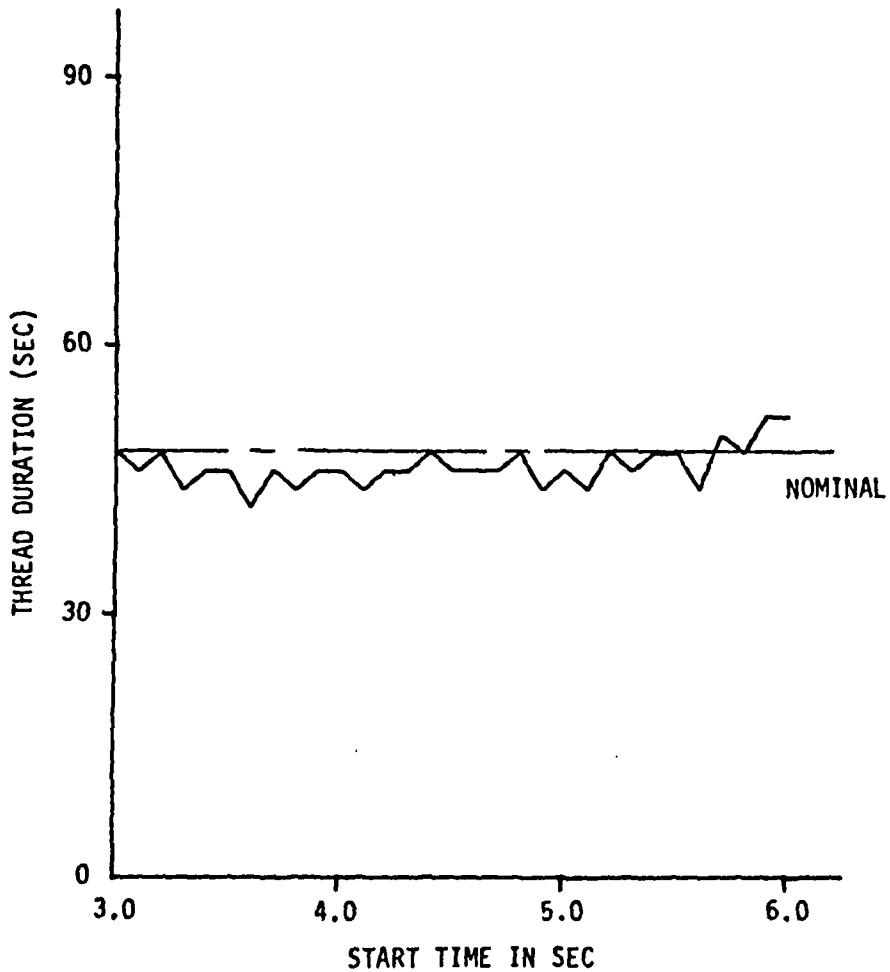


Figure 3.6-5. Normal Track Thread During Intercept Planning

Port-to-Port Response Time

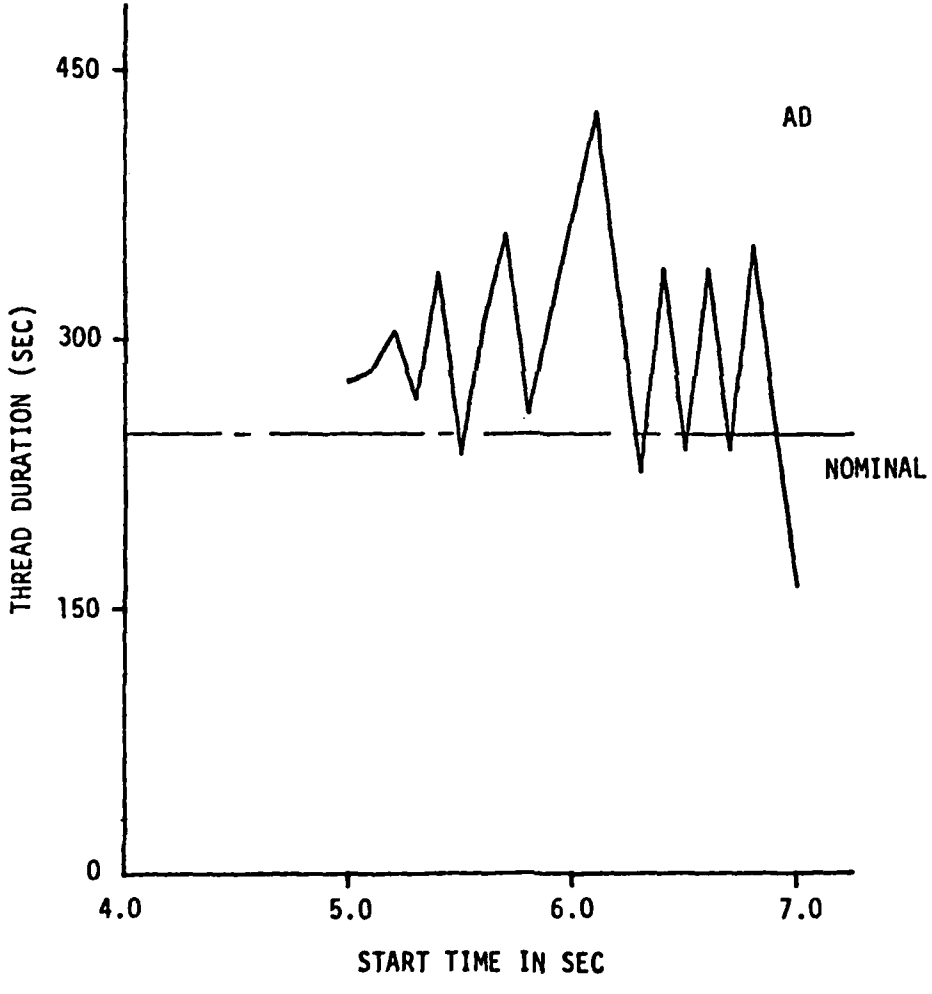


Figure 3.6-6. Active Discrimination Thread
Port-to-Port Response Time

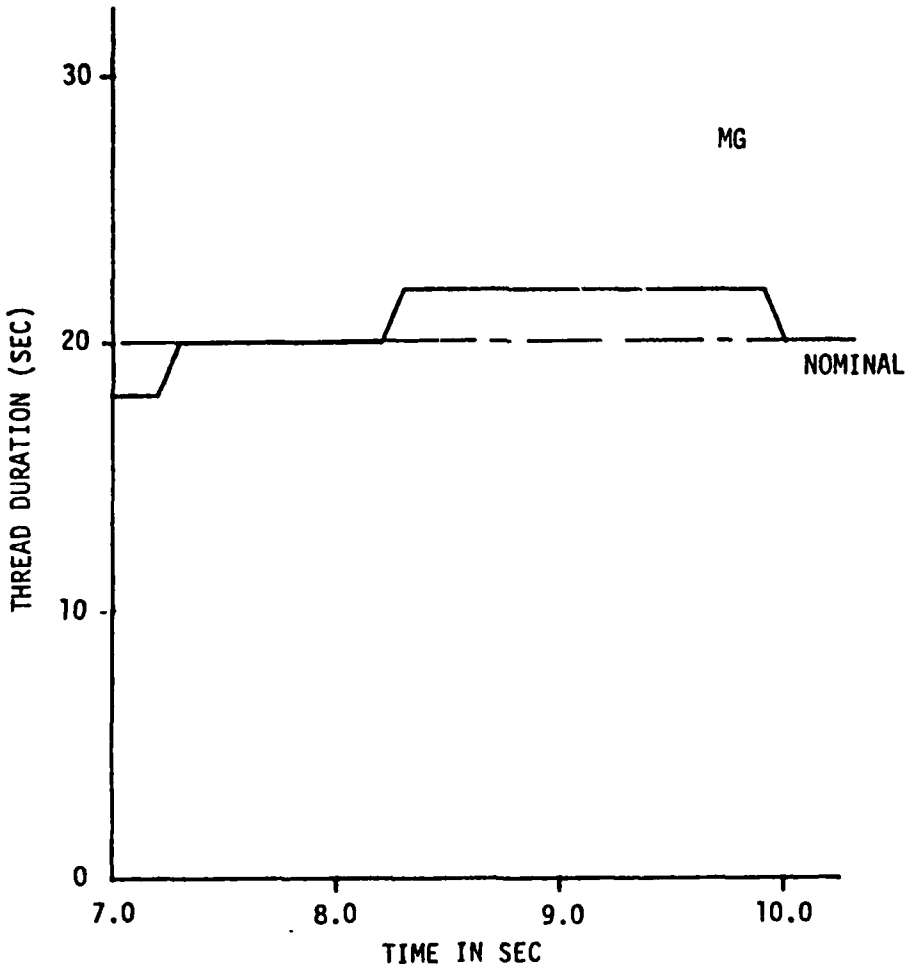


Figure 3.6-7. Interceptor Guidance and Track Thread Port-to-Port Response Time

3.6.2 Process Loading Analysis

PAES has the capability to record flows through any set of queues in the process. This capability was exploited to provide an analysis of the process loading on various portions of the process associated with a scenario, independently of any specificaton of hardware architecture. We found this information to be extremely valuable in synthesizing the hybrid architecture. It is interesting to observe that in this problem maximal loading can be determined accurately by PAES. This is because the thread response times are the duty cycle goals required for processing the returned radar pulses. This is significant because the tracking filters will be designed to perform best at a nominal update rate. They will give the needed track accuracies. Higher update rates give unneeded improvements in tracking accuracy. Thus by setting thread delays (by inserting appropriate delays in the "DTF" primitives) we can observe the characteristic system loading that is determined by the selected scenario. This was the approach taken.

A second observation addresses the scenario. There is a bound on the number of RV's that can be productively used against a single BMD site. The fratricide effect provides one limit of the RV density to be expected. Cost benefit analysis provides another. Thus, we can bound the number of RV's this is required to be addressed in an attack occurring over several seconds. The number of ghosts, fragments, and decoys associated with a set of RV's can be estimated with good accuracy. The question remains, what temproal distribution of this set produces the heaviest loading on the site processor? The answer depends on the architecture of the site processor. In a poor architecture, loading on one thread might produce a bottleneck in the processing of a second thread. However, assuming a responsive architecture (specifically, one in which no queueing for processing occurs) the loading is maximized by a single dense wave of RV's occuring within a few milliseconds and representing a simultaneous launch. This conclusion follows from that the fact that, under the stated assumptions, the loading is a linear functional of the input S/V time history. Therefore, maximum peak loading on any portion or all of the process is obtained with a maximally peaked S/V time history.

In conformity with this observation, we have used, as a baseline, a scenario with closely grouped RV's to determine the inherent processing loading.

An issue of interest in characterizing the underlying process is the sensitivity of the process to variations in the scenario. We have examined this by running the PAES process model with a second scenario that has two RV waves separated by approximately one second.

Samples of the loading - the TI and OT2 thread for the single and double spike scenario - are provided in Figures 3.6-8 and 3.6-9. These results were run with a simulation cycle time of 2ms. Loading is presented for a smoothed value over a 60 ms interval. These smoothed plots represent the realistic loading in an architecture with some buffer elasticity (perhaps 5 msec of data) between functions.

A summary of the system loading generated by the single spike scenario is given in Figure 3.6-10. Interesting features here are:

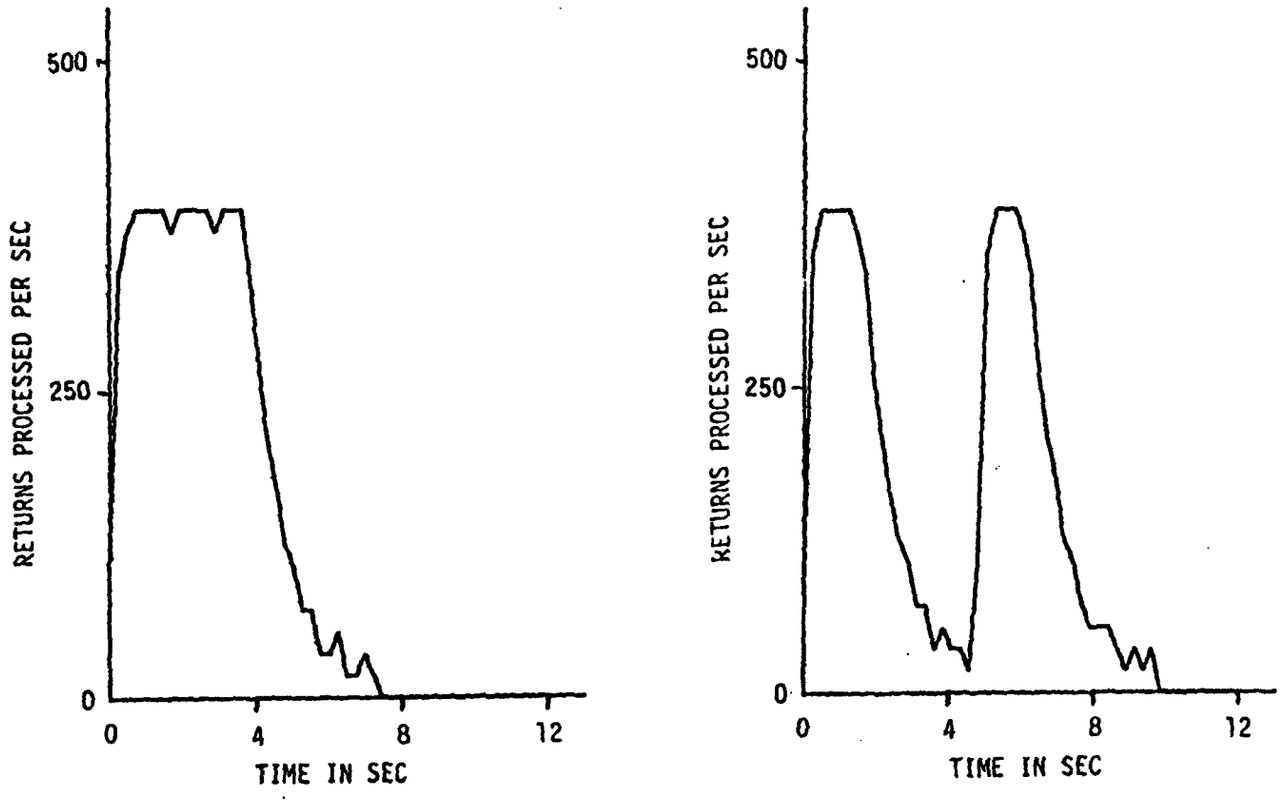
(1) Contributors to the total peak loading are TI, OT1, OT2, OT3, OT4, RRA, MICRO and MACRO. The principal contributors are, in order, OT1, then OT2. Not contributing are SV, OT5, IP and IC.

(2) The loading patterns for RRA, MICRO, MACRO and all others combined are very similar.

3.6.3 Architectural Implications

The response time requirements (Table 3.5-1) and the loading assoicated with a scenario have implications for all hardware architectures which might be considered. These implications are examined here.

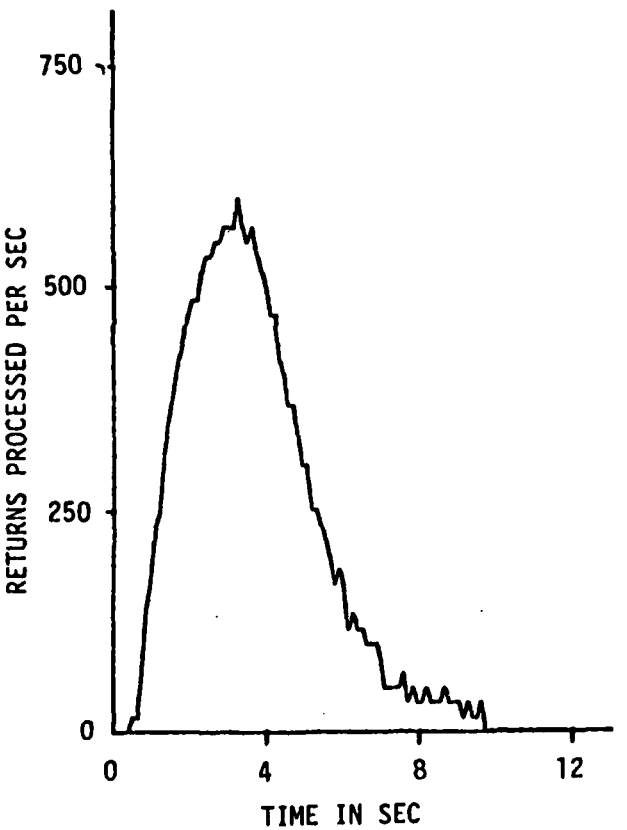
Thread response times lead to a processing requirement in terms of a minimum cpu power (instructions per sec). Specifically, if the thread response time requirement is T and the thread processing requirement per



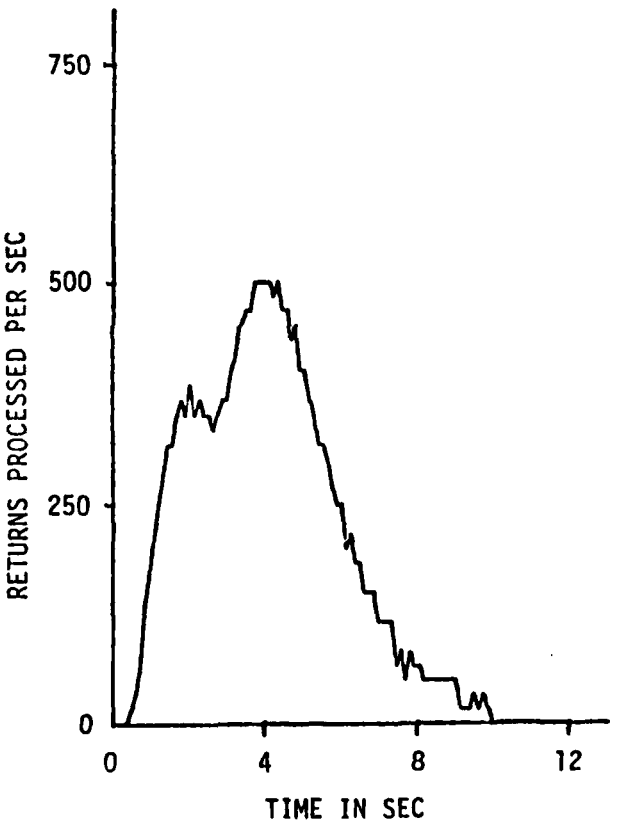
(a) Single-Spike Scenario

(b) Double-Spike Scenario

Figure 3.6-8. TI Returns Processing Load



(a) Single-Spike Scenario



(b) Double-Spike Scenario

Figure 3.6-9. OT2 Returns Processing Load

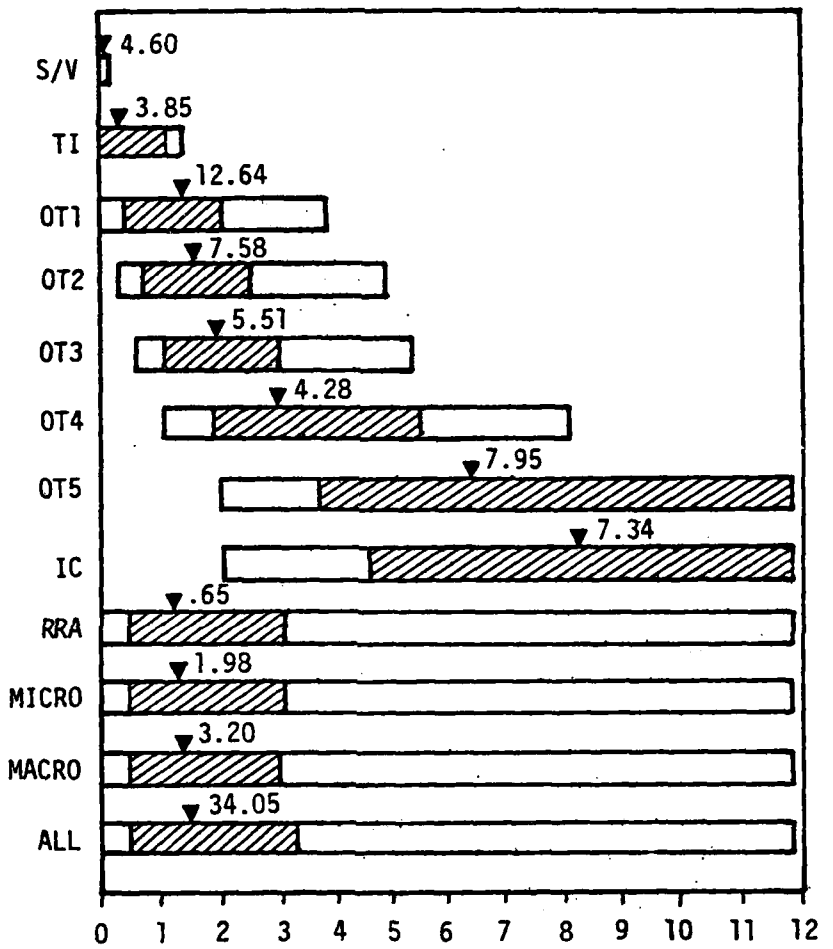


Figure 3.6-10. Load Phasing for the Single-Spike Scenario

work unit is p, then the minimum cpu power applied must be p/T.

Thread response-time-driven cpu requirements were calculated for each thread, with results given in Table 3.6-1. Opposite each thread name, the "total thread EMLI" is given. This quantity is determined from the instructions per work unit and instruction type for each PRIMITIVE on the thread, excluding those related to the radar interface functions of radar returns assimilation, micro and macro scheduling. Specifically,

$$\text{EMLI} = \sum_{\substack{\text{primitives} \\ \text{on thread}}} [1.064 \times \underset{\text{instructions/w.u.}}{\text{floating point}} + .532 \times \underset{\text{instructions/w.u.}}{\text{integer}}]$$

(The rationale for this is provided in Section 4.1) To account for the processing time involved in these radar related functions and for communication delays, the response time available was reduced by 15 msec for most threads, and by 10 msec for the interception threads, OT5 and IC. In the latter cases, it is assumed that priority techniques can be applied, especially in scheduling, to minimize delays and thereby make the indicated processing times available.

The derived cpu requirements are in the range of .10 to .35 MIPS for all threads, except for OT5 and IC. There the requirements are .82 and 1.22 MIPS, respectively.

Thread loadings corresponding to a particular scenario lead to a second set of cpu requirements, but in this instance, the requirement is for total cpu power available whether provided by one or by several machines. To derive these requirements, the "total thread EMLI" is multiplied by the peak workunit rate experienced by that thread in the chosed scenario. Results for both the single spike and double spike scenario are presented in Table 3.6-2. By comparing these results with those of Table 3.6-1, it can be seen that

- the loading-driven cpu requirements exceed the response-time-driven cpu requirements for every thread. This fact implies

Table 3.6-1. Thread Response-Time-Driven Cpu Requirements

Thread	Total Thread EMLI*	Response Time Requirement including RRA, MICRO, & MACRO (msec)	Allocated Processed Time Excluding RRA, MICRO, & MACRO (msec)	Required Power (MIPS)
SV	6 577	50	35	.19
TI	10 058*	50	35	.29
Final TI	9 044**	50	35	.26
OT1, OT2, OT3, OT4	12 236	50	35	.35
OT5	12 236	25	15	.82
IP	7 714	100	80	.10
IC	12 236	20	10	1.22

* 1.064 x floating MLI + .532 integer MLI, excluding radar functions

** TI/OT correlation is not on the final TI thread

*** based on lower response time requirement

Table 3.6-2. Thread Loading-Driven Cpu Requirements

Thread	Total Thread EMLI	Peak Loading					
		Single-Spike			Double-Spike		
		time (msec)	w.u./ sec	MIPS	time (msec)	w.u./ sec	MIPS
SV	6 577	60	700	4.60	60	367	2.41
TI	10 058	300	383	3.85	200-1380	383	3.85
Final TI	9 044	420	133	1.20	300-1400	133	1.20
OT1	12 236	1000	1033	12.64	1680	1017	12.44
OT2	12 236	1620	600	7.58	1980	500	6.12
OT3	12 236	1980	450	5.51	2500	383	4.69
OT4	12 236	3000	350	4.28	3500	317	3.88
OT5	12 236	6500	650	7.95	7400	517	6.33
IP	7 714	3000-7000	17	0.13	3000-8000	17	0.13
IC	12 236	8400-10200	600	7.34	10500	583	7.13

that parallel processing would be useful, since smaller (lower cpu) machines could then be used.

Close examination of the loading near the time of the peak total loading is useful. Figure 3.6-11 provides details in the relevant interval.

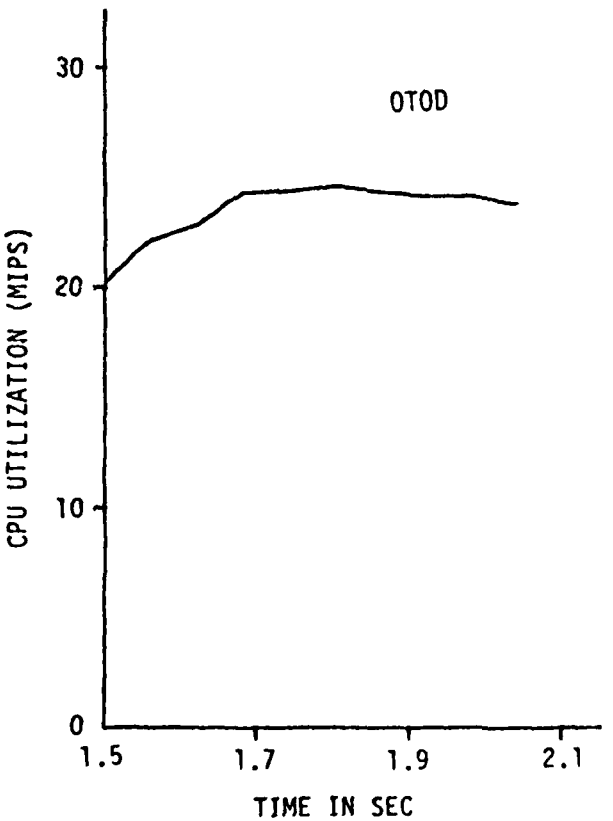
- The separate radar interface functions, RRA, MICRO and MACRO and the combined thread oriented processing each peak at very nearly the same time. Therefore, allocation of these four groups of processing to distinct NODES will not require any more total cpu power than in the minimum cpu environment of one central NODE.

- The peak loading is determined by the pattern of the TI, OT, (except OT5), OD and object correlation (OBJCOR) functions since the radar-interface function loadings are fairly uniform in the relevant interval and the SV, OT5, IP, and IC do not contribute at all. As noted previously, OT1 is the principal contributor. In order to minimize cpu requirements, process design should therefore be directed to those functions noted as determining the peak load.

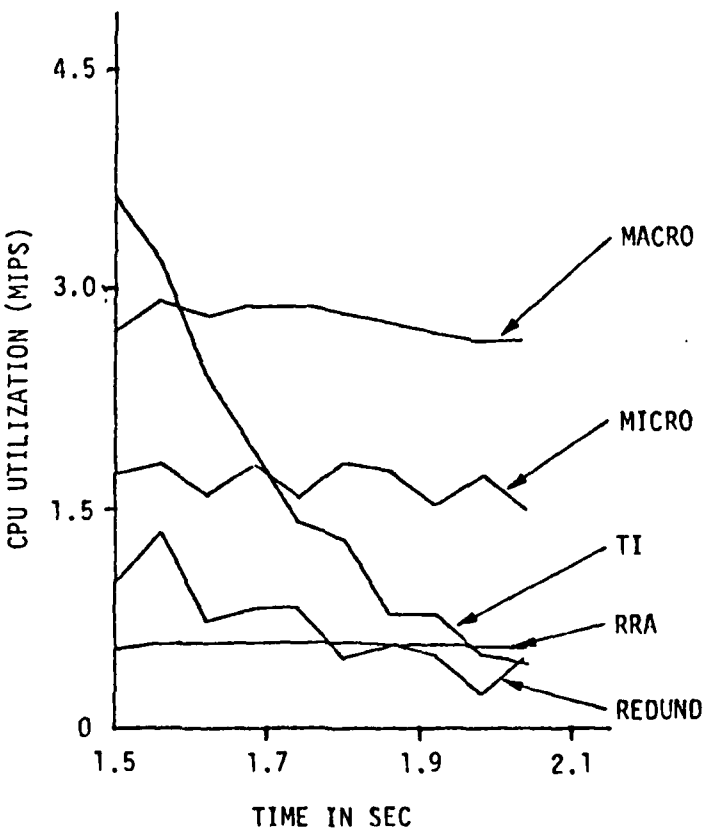
In summary, the following features of the process are significant for hardware architecture design:

Feature 1: Response time requirements are much less than total loading requirements. Thus, a parallel architecture is appropriate.

Feature 2: Bus response, memory access, and operating system overhead strongly impacts the processing speed required to do the OT5 and IC functions.



(a) Obejct Track and Discrimination Threads



(b) Other Relevant Threads

Figure 3.6-11. Thread Contributions to Peak Loading (Double-Spike Scenario)

Feature 3: The thread processing requirements for almost all functions are under 0.5 MIPS. Thus, advanced high speed processing architectures do not have to be used for these functions.

Feature 4: OT1 loading uses over 50 percent of the processing power required during the maximum loading of the system.

4.0 EVALUATION OF NETWORK ALTERNATIVES

The principal purpose of this study has been to synthesize and evaluate an "optimal" computer network architecture for supporting the BMD data processing subsystem. To arrive at a sound basis for this endeavor, the process has been thoroughly examined, as documented in the previous section. Further, previously developed approaches to architecture were assessed. In this section, two such architectures are examined - Centralized and Thread. The remainder of the discussion is devoted to the description of and evaluation results for the Hybrid Architecture that was synthesized.

4.1 Evaluation Procedure

Before addressing the evaluation results themselves, we describe here the procedure that was used in arriving at evaluations of network alternatives. An overview of this procedure is illustrated in Figure 4.1-1.

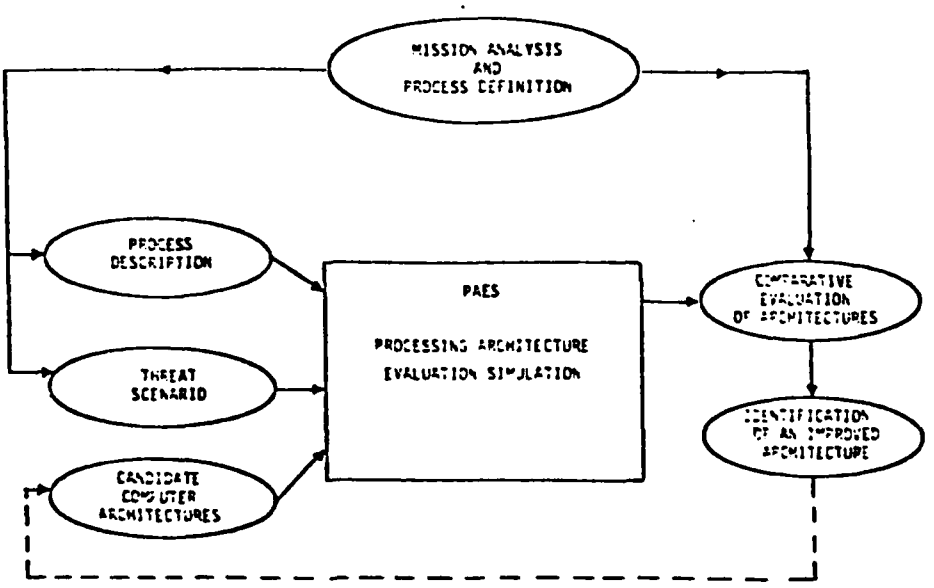


Figure 4.1-1. Network Analysis Procedure

Mission analysis and process definition, as described previously in Section 3, provided a formal description of the process, scenarios and requirements. These served as two of the major input segments to the Processing Architecture Evaluation Simulation (PAES). In the instance of the previously developed candidate computer architectures - Centralized and Thread - available descriptions were used to identify necessary data to define architectural inputs for PAES.

In the instance of the Hybrid Architecture, process loading analysis (Section 3.6) and results from the assessments of the two other architectures were used to synthesize the design. The corresponding architectural inputs for PAES were then identified.

In using PAES, two aspects of the characterization of architectures are worthy of note: (1) machine speed, and (2) representation of architectural effects.

In the VERAC model of the PRIMITIVES, algorithm size is set in terms of MLI's with the followng first cut assumptions:

Assumption 1: The CDC 7700 operates at 12.87 MIPS when executing a specified "BMD mix" of software.

Assumption 2: Because of the optimization in the design of the control sections of the 7700, fixed point instruction mixes are executed about as fast as floating point instructions, i.e. at 12.87 MIPS.

Assumption 3: The full process simulated on PAES has the same instruction mix as the "BMD mix" used for bench marking.

These three assumptions allow use of McDonnell Douglas-generated timing data for the determination of algorithm size.

$$\text{Algorithm size (in MLI)} = 12.87\ T_a,$$

where: T_a is the execution time for the algorithm code set as measured in the CDC environment (in seconds).

A further set of assumptions allow us to relate algorithm size to the performance parameters appropriate for the distributed processing environment:

Assumption 4: Smaller computers perform at half the rate for instruction mixes rich in floating point operations (filtering, correlation) compared with mixes that are predominately fixed pont (e.g. formatting, control logic).

Assumption 4 indicates a skew in total computational time required towards the time required by floating point algorithms as we drop from very large computers to smaller computers. A transform paramter x is used to adjust for this skew:

x = Equivalent MLI, (EMLI) for the floating point instruction mix.

0.5 x = Equivalent MLI for the fixed point instruction mix.

Parameter x has been determined by holding constant the total number of equivalent instructions executed in a scenario for the CDC 7700 implementation and the distributed processing implementation. Experimental evaluation of x using simulation PAES has given:

$$x = 1.064$$

Note, as an aside, that this experiment has given a measure of the mix of floating-point type and fixed-point type algorithm instructions executed during a scenario. Let f be the fraction of executed instructions that are of the floating-point-mix type and k be the number of instructions. Then

$$1.064fk + .532\ (1-f)k = k,$$

from which we find,

$$f = .88.$$

Thus, 88 percent of the site defense instructions executed are in algorithms of the floating-point-type.

We have used the equivalent CDC 7700 MLI, designated EMLI, as the performance measure for architectures

$$\begin{bmatrix}\text{1 algorithm step} \\ \text{of fixed-point type}\end{bmatrix} = 0.532 \text{ EMLI}$$

$$\begin{bmatrix}\text{1 algorithm step} \\ \text{of floating-point type}\end{bmatrix} = 1.064 \text{ EMLI}$$

If it is assumed that the BMD instruction mix is similar to a standard benchmark mix such as the Wheatstone, then the results reported herein, in EMLI, can be held to be equivalent to benchmarked instructions (e.g., for example, assuming the CDC 7700 would test at 12.87 MIPS using the Wheatstone benchmark). However, becuase of the 88 percent content of arithmetically oriented code in the BMD, the standard benchmarking would be expected to give a higher number (e.g., 13.5 - 15 MIPS for the CDC 7700 by the Wheatstone). We will present results with EMLI because of the lack of benchmarks that relate the BMD code to a standard. The reader may interpret these results as benchmark MIPS, but the above caveat should be kept in mind.

Computer architectures, with associated communication buses, have finite capabilities for processing and transferring data. In PAES, these effects are represented by architectural effects models corresponding to

- operating systems
- bus protocols
- resource access (e.g., shared memory).

PAES was developed for this effort to have an essential but limited capability for representing these effects. In particular, a simple, "fixed allocation" operating system model was implemented which assigned a fixed processing power to each PRIMITIVE thorughout the scenario. Generally, the allocation to each PRIMITIVE was sufficient to avoid

queueing. Thus, we did not attempt to identify congestion effects of realistic, finite computing resources. Rather, we measured the cpu requirement necessary to avoid queueing.

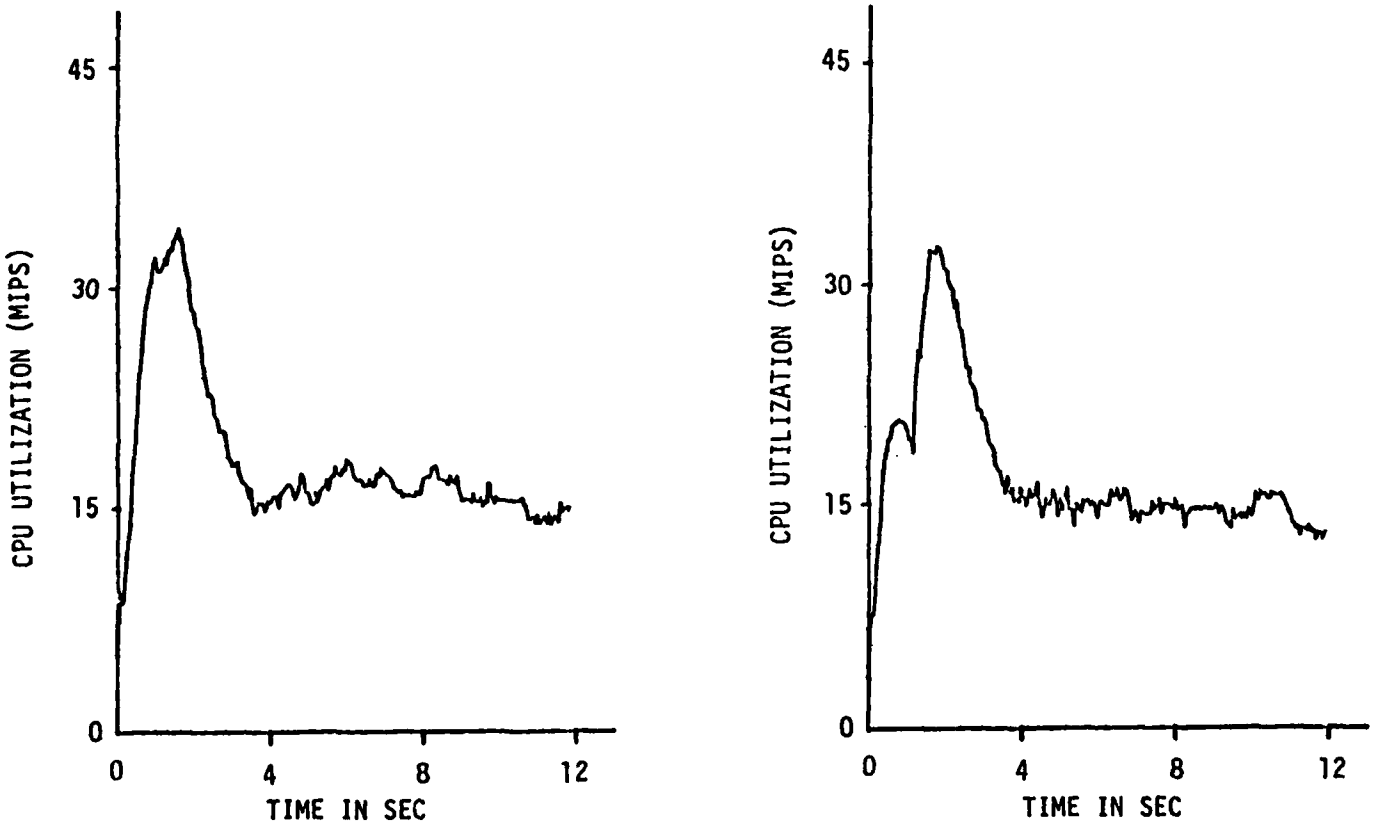
Our assessment of distributed processing alternatives lead us to believe that delays in data transfers would be substantially less than the processing times on threads. Correspondingly, we did not implement delay models for bus protocols or resource access, and therefore developed no measures from PAES for these effects. In the discussion of the Hybrid Architecture, however, we do address bus communication delays through simple analysis.

It is to be emphasized that PAES is capable of accounting for delays caused by queueing and architectural effects. Realistic evaluation of such effects only requires a modest further development of the associated architectural effects models.

4.2 Evaluation of the Centralized Architecture

Orginal development of the BMD data processing subsystem was based upon implementation on a single large computer - the CDC 7700. Our first architectural evaluation was of a representative Centralized Architecture. The key feature is the sharing of a single cpu by all processing functions. In this respect, it is noted that this deviates from the two-cpu architecture of the CDC 7700. Further, no attempt was made to represent other CDC 7700 environment effects, such as the use of TOS (Tactical Operating System) and the data and instruction code transfers between the LCM (large core memory) and each of the two SCMs (small core memories).

Results are displayed in Figure 4.2-1 for the single-spike and double-spike scenarios. There, 60 ms smoothed cpu utilization requirements are displayed. Peak requirements are seen to be 34.05 MIPS for the single-spike scenario and 32.50 MIPS for the double-spike scenario.



(a) Single-Spike Scenario

(b) Double-Spike Scenario

Figure 4.2-1. cpu Utilization for Centralized Architecture

The centralized environment has the advantage of having the full cpu processing available to all processing needs. With a responsive operating system (such as TOS), processing power can be allocated as needed in the particular scenario. This flexibility is had at the cost of a complex operating system which can introduce very substantial overhead (as is the case in the CDC 7700 implementation). To provide some means of comparison with the other architectures, we assume an overhead cost of 40 percent, so that we obtain the total cpu requirements as indicated in Table 4.2-1.

Table 4.2-1. Cpu Requirements for the Centralized Architecture

Scenario	Application Requirement (MIPS)	40 percent Overhead (MIPS)	Total cpu Requirment (MIPS)
Single-Spike	34.05	13.62	47.67
Double-Spike	32.10	12.84	44.94

4.3 Evaluation of the Thread Architecture

The first significant development of a distributed computer network alternative to the CDC 7700 architecture was the Thread Architecture design made by McDonnel-Douglas [3]. The unit-level portion of this architecture is displayed in Figure 4.3-1. There, 17 computers and the major control flow interactions are indicated.

In Figure 4.3-2, the allocation of PRIMITIVES to the nine functional NODES of the thread architecture are indicated. (The details, i.e., specification of PRIMITIVES, can be seen by examining Figure 3.3-2 which is a larger version of Figure 4.3-2 with PRIMITIVE names.) It will be noted that two NODES contain more than one computer: NODE TI contains two computers (Nos. 5 and 6); and NODE OTOD contains eight computers (Nos. 8-15). In using PAES, we measure NODE cpu requirements and leave details of local NODE operating systems for future analysis.

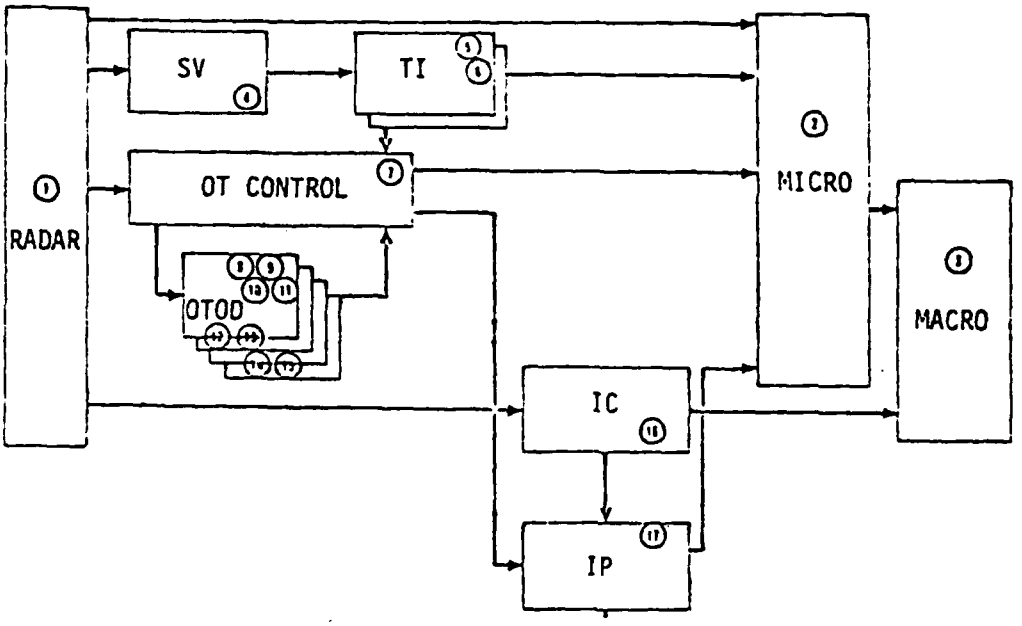


Figure 4.3-1. Thread Architecture

Results of PAES evaluations for the two scenarios are given in Table 4.3-1. Summing the peak NODE cpu requirements yields requirements of 51.83 MIPS for the single-spike scenario and 45.01 MIPS for the double-spike scenario. We have also attempted to account for operating system overhead here. Since the number of functions supported within each NODE is much smaller than in the centralized environment, we have assigned an overhead of 20 percent in order to arrive at total cpu requirements. These requirements are also indicated in Table 4.3-1.

Further detailed results in the form of sampled and smoothed cpu requirements for each NODE and each scenario are provided in Appendix C.

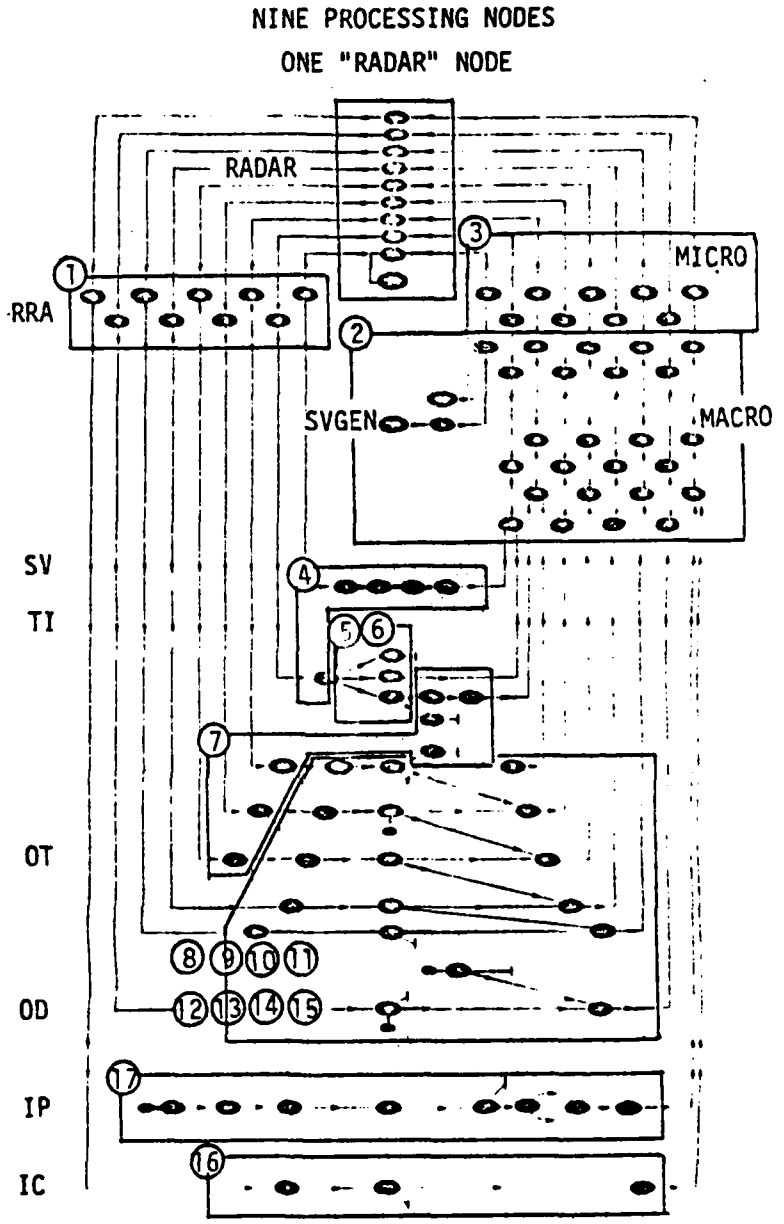


Figure 4.3-2. Allocation of PRIMITIVES to NODES of the Thread Architecture

Table 4.3-1. Cpu Requirements for the Thread Architecture

(a) Single-Spike Scenario

NODE	Application Requirement (MIPS)	20 percent Overhead (MIPS)	Total cpu Requirement (MIPS)
RRA	0.65	0.13	0.78
MACRO	3.20	0.64	3.84
MICRO	1.98	0.40	2.38
SV	4.12	0.82	4.94
TI	3.72	0.74	4.46
REDUND	1.43	0.28	1.71
OTOD	28.92	5.78	34.70
IC	7.45	1.49	9.94
IP	0.36	0.07	0.43
TOTAL	51.83	10.34	62.17

(b) Double-Spike Scenario

NODE	Application Requirement (MIPS)	20 percent Overhead (MIPS)	Total cpu Requirement (MIPS)
RRA	0.59	0.12	0.71
MARCO	2.91	0.58	3.49
MICRO	1.82	0.36	2.18
SV	2.24	0.45	2.69
TI	3.63	0.73	4.36
REDUND	1.35	0.27	1.62
OTOD	24.66	4.93	29.59
IC	7.45	1.49	9.94
IP	0.36	0.07	0.43
TOTAL	45.01	9.00	54.01

4.4 Design and Evaluation of the Hybrid Architecture

In this section, we present in detail a recommended architecture, which we shall refer to as the Hybrid Srchitecture due to the mixture of types of network architectural elements present. This architecture was developed by VERAC as a result of its analysis of the process to be supported, and its assessment of other architectures. Again, PAES evaluation of cpu requirements are presented. Further, in this instance, we address additional design details related to the organization of control, bus loadings and database management.

Figure 4.4-1 presents this Hybrid Architecture. The principal of the design is to use a parallel computer architecture with a central memory for the data base for the bulk of the processing, and to use special purpose processors to handle control and scheduling.

4.4.1 Functional Partitioning

Figure 4.4-2 presents the functional partitioning for the system. The RRA and SCHEDULER functions are set in distinct nodes. A parallel set of THREAD processors is represented as the AGGPRO NODE. Two additional processor types are included in the system model of Figure 4.4-1: the Unit Resource Module (URM) and the Memory Processor (MEM). These have not been sized. However, their functions are described in following subsections.

4.4.2 PAES Evaluation of CPU Requirements

Results of PAES evaluations for the two scenarios are given in Table 4.4-1. Summing the peak NODE cpu requirements yields requirements of 34.10 MIPS for the single-spike scenario and 32.55 MIPS for the double-spike scenario. Note that these figures are only slightly larger than the results for the Centralized Architecture. As was observed in the loading analysis of Section 3.6, the organization of processing involved in this Hybrid Architecture retains almost all of the

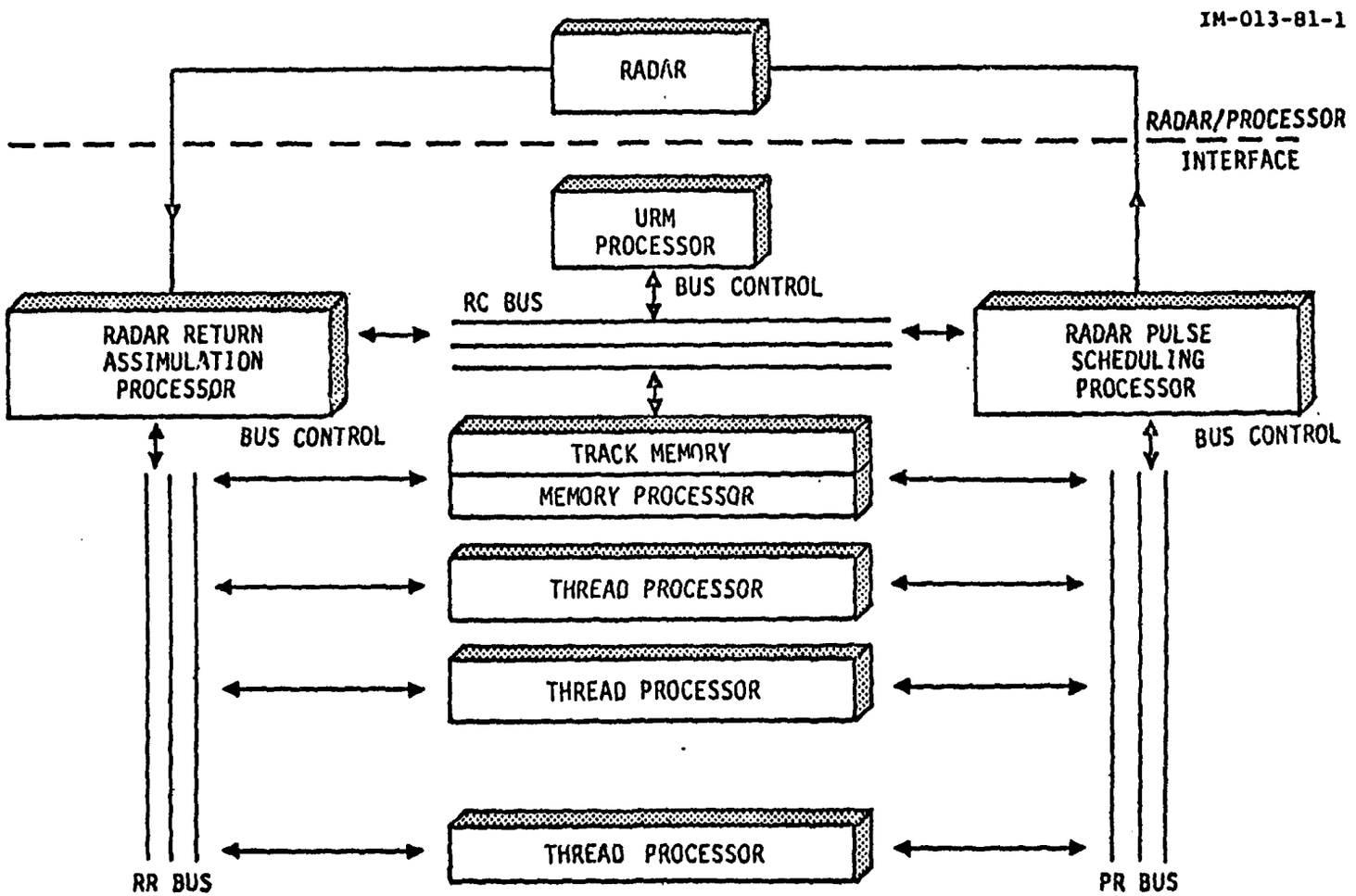


Figure 4.4-1. Recommended Architectural Approach for the BMD Site Defense Distributed Processor

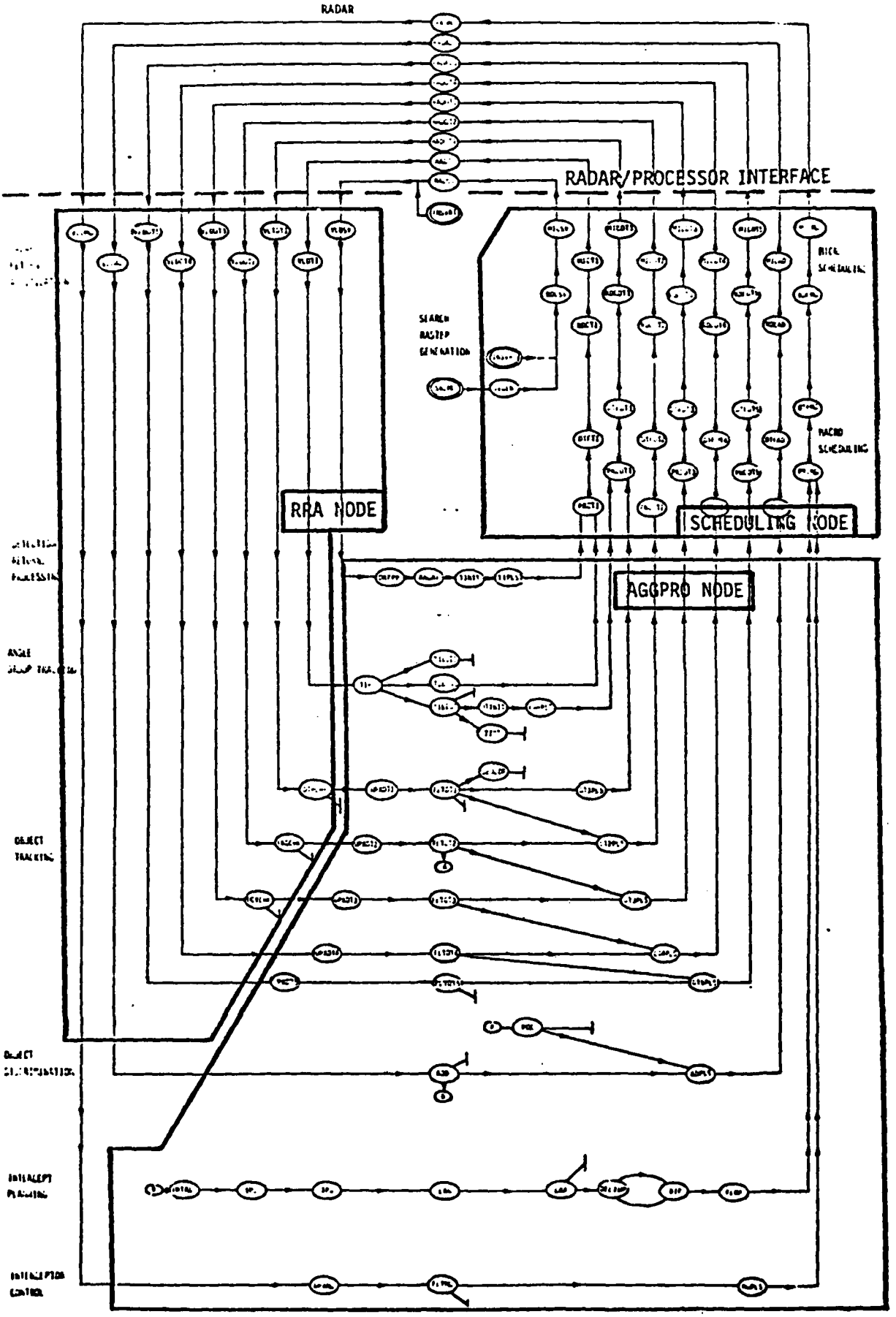


Figure 4.4-2. Allocation of Primitives to Nodes of the Hybrid Architecture

Table 4.4-1. Cpu Requirements for the Hybrid Architecture

Single-Spike Scenario

NODE	Application Requirement (MIPS)	Other Loading (MIPS)	Total cpu Requirement (MIPS)
RRA and CONTROL	0.65	0.50	1.05
SCHEDULER	5.18	0.20	5.38
AGGPRO (thread processors)	28.27	2.83	31.10
TOTAL	34.10	3.53	37.63

Double-Spike Scenario

NODE	Application Requirement (MIPS)	Other Loading (MIPS)	Total cpu Requirement (MIPS)
RRA and CONTROL	0.59	0.50	1.09
SCHEDULER	4.73	0.20	4.93
AGGPRO (thread processors)	27.23	2.72	29.95
TOTAL	32.55	3.42	35.97

efficiency of shared processing associated with a single cpu architecture. The critical assumption is that the CONTROL function can be designed to efficiently use the multi-processor AGGPRO NODE.

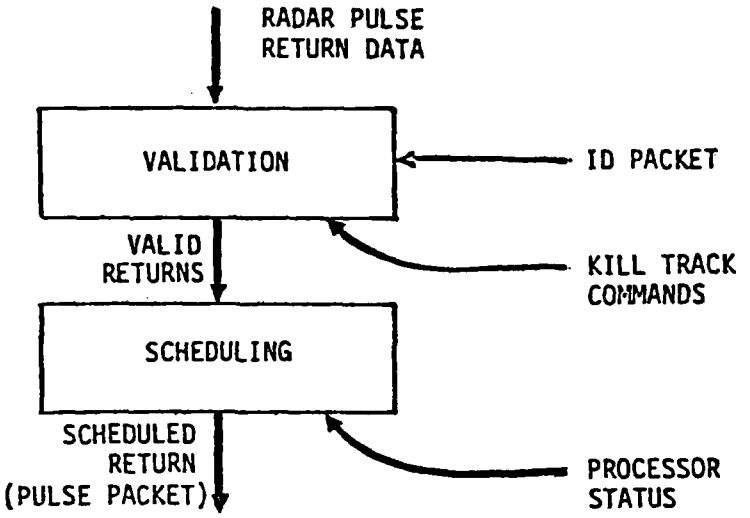
Also indicated in this table are "Other Loading" cpu requirements. In the RRA and CONTROL NODE, this other loading corresponds to the control of radar return pulse processing, as described further in Section 4.4.3. In the SCHEDULER NODE, the other loading is associated with control of the PR bus, described further in Section 4.4. And for the AGGPRO NODE, we have assigned an overhead of 10 percent. This lower figure (compared to 20 percent for Thread Architecture nodes) is related to the very simple operating systems required for thread processors. All the complex "tasking" is taken care of in the RRA and CONTROL node.

4.4.3 RRA and CONTROL NODE

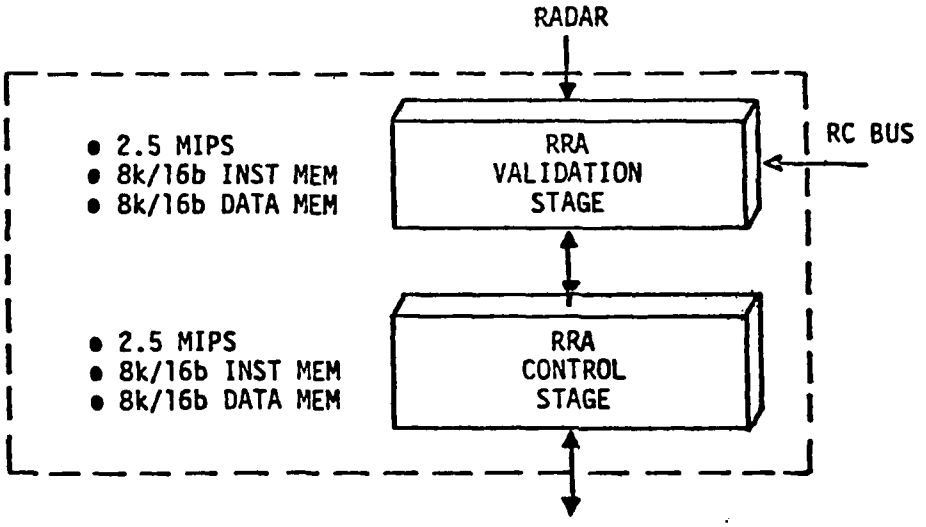
The RRA and CONTROL NODE provides the validation check of a radar return (VLD.). Additionally, this processor schedules use of the local processors and cancels local tracks, (OTRCHK, FRGCHK, DCYCHK). Figure 4.4-3 depicts the data flow and processor structure for the recommended approach.

When a pulse transmission request is sent to the radar by the SCHEDULER, the SCHEDULER also forwards a packet of descriptive data to the RRA processor. This packet contains the related track or group ID (it is named the "ID Packet") and the pulse-type data. The RRA processor uses the track ID information to search a list of track kill requests. If a track is to be killed, this information is put in the ID Packet in preparation for processing the pulse that is to be received. The first function performed by the RRA on a return pulse is the validity check against the ID Packet (VLD.). The ID Packet is appended to the return information at this point.

The second function performed by the RRA on a packet is the assignment of the return to a THREAD Processor for processing. The assignment algorithm uses a PROCESSOR STATUS list kept by the RRA processor.



(a) Data Flow



(b) A Special Purpose Two-Stage Processor Implementation

Figure 4.4-3. The RRA and CONTROL NODE Structure

THREAD Processors are not, in general, assigned to specific function types or to specific tracks. These processors are viewed as being structured to perform any of the functions contained in the THREAD NODE. the RRA Processor selects a currently idle THREAD processor and sends the pulse return with appended ID to this processor. The packet so transmitted is called a "Pulse Packet". There are two possible exceptions to this treatment of a pulse return that can be considered.

We have included as part of the RRA processing, in our approach, the interdiction mechanism required to drop a track. The kill request information is simply included with the ID and appended during VLD. processing. During scheduling, a kill request in the packet is noted and the RRA responds by sending off a packet indicating that the track has been killed (i.e. not forwarded for processing). This approach integrates the track kill mechanism without requiring a new or special algorithm stage.

The second exception to be considered is the case for returns that are associated with intercept (OT5 tracking and IC). This processing is sensitive to memory access times. Thus, it may be appropriate to transfer to, and maintain the target and interceptor tracks in, the memory of a THREAD processor. In this case, the processor identity would be carried in the ID Packet. The RRA processor would note the assignment in the ID appendage to the return and would route the return accordingly.

Fault tolerance is a major factor in the processing architecture recommended. The parallel THREAD processors can perform self-test as assigned by the Scheduler. Processor status determined by the self-tests can be returned to the RRA Processor and entered along with the measure of loading into the THREAD Processor Status Table. Also, fault detection by a THREAD Processor would be reported in this way.

This structure allows faulty THREAD processors to be removed in real-time without significant impact or overhead on the ongoing system performance. Since the main segment of the system is the set of THREAD processors, this structure relieves much of the cost and risk associated with designing extremely reliable individual processors.

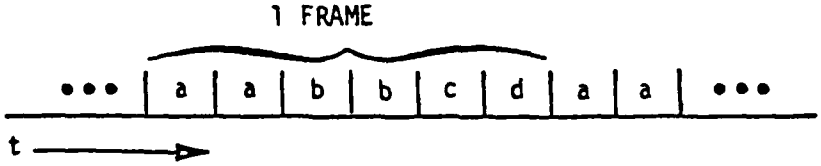
4.4.4 The RR BUS

The RR BUS transports the pulse return data ("Pulse Packet") from the RRA Processor to the assigned THREAD Processor. The MEM Processor also picks up the Pulse Packet. The RR Bus transports track data from MEM to the THREAD Processors as required. Additionally, control and status data is returned via the RR BUS to the RRA Processor. The RR BUS bandwidth is determined by the maximal pulse rate. The principal loads on the RR bus are the pulse return packets and the track records used by the thread processors. Each packet and track record is taken to be 150 32-bit words, i.e., 4800 bits. The bandwidth requirement shown in Table 4.4-2 is based on these data sizes. This BUS could be easily implemented with a 32 bit wide, 1Mwps architecture. This architecture would easily support a radar rate of as high as 10,000 pps.

Table 4.4-2. Bandwidth Requirements at 2000 pps for the RR BUS

Function	Rate
Pulse Returns	9.6 Mbps
Track/Group Packets for Filtering	9.6 Mbps
Correlation Track Packets (10 correlation candidates /request σ 100 requests/sec.)	4.8 Mbps
Status Reporting (320 b/report)	.64 Mbps
Total	24.64 Mbps

The RR Bus is under the RRA Processor control. A simple protocol is to have capacity assigned in TDMA fashion for pulse returns and additional capacity available by demand access for correlation data transfer. This structure is shown below.



a: Pulse Packet (150w)
b: Track/Group Packet (150w)
c: Correlation Packet (150w)
d: Status/Request Return Set (10w/Thread Processor)

This simple protocol can work effectively because of the very uniform rate of radar returns. This protocol would provide a transfer in under 1 ms for each packet.

4.4.5 THREAD Processor

The THREAD Processors are collectively given the capability to perform all of the functions in the AGGPRO NODE as specified in Figure 4.4-2. These are:

(1) S/V Processing
(2) TI Processing
(3) OT Processing
(4) Correlation/Discrimination
(5) Intercept Planning
(6) Intercept Control.

The total instruction code for the thread functions is estimated to be 160,000 32-bit words. If each THREAD processor has instruction and data storage capability of 256,000 words, every THREAD processor could have the capability to perform every thread processing function. If

less memory is available, it would be necessary to allocate portions of the thread functions to THREAD processors. By taking note of the load phasing as depicted in Figure 3.6-10, it should be possible to provide an allocation which requires only slightly more total cpu capacity than the minimum necessary if all THREAD processors were fully capable.

The specific function or "task" to be performed by the processor is determined by the RRA when it schedules return processing and other activities. Typical tasks are the filtering and processing associated with a TI or OT return. Other tasks assigned are correlation, intercept planning, and intercept control.

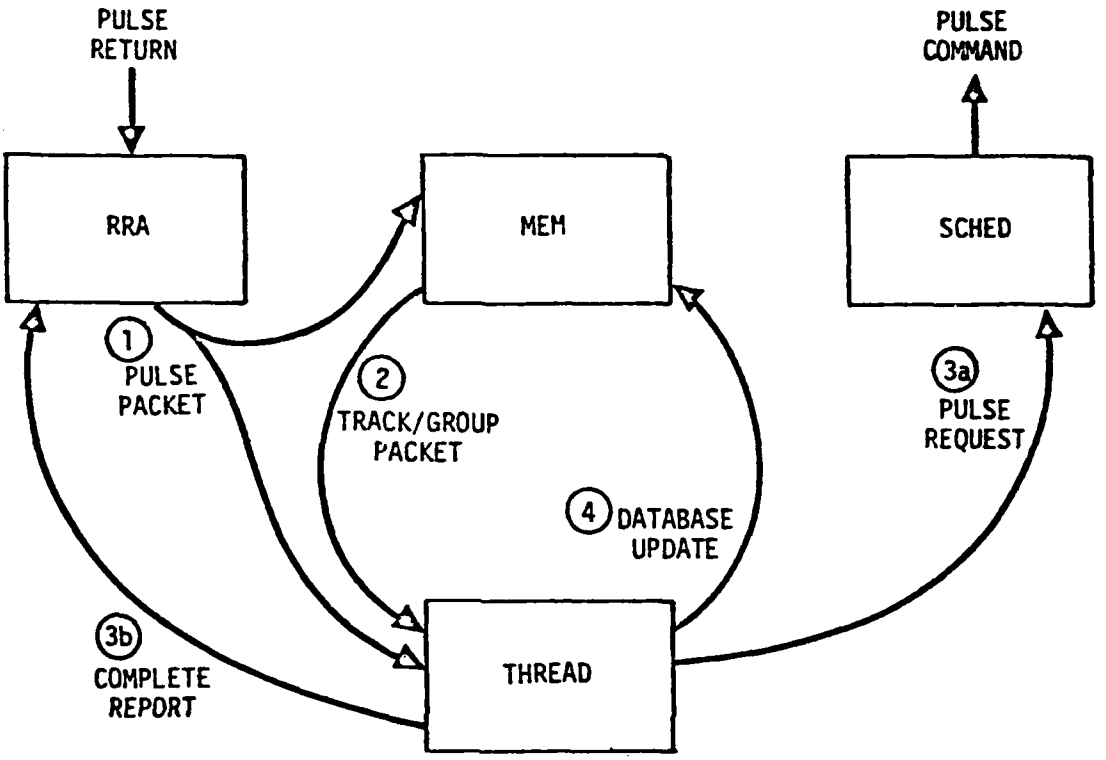
Pulse packets are sent to the assigned processor by the RRA. These packets are also received by the Memory Processor, MEM. The MEM processor retrieves from the track data base the track information used in filtering. This data is forwarded, as a "Track-Packet" or "Group-Packet", on the RR BUS to the assigned THREAD processor. This scheme allows the memory access to be carried out while the THREAD processor is unpacking the Pulse Packet. Figure 4.4-4 shows the data flow that is associated with pulse return processing.

The THREAD Processor processes the pulse return data. Typically a new pulse request is generated (.PLS) as well as an update made to the track/group data base. This is shown in Figure 4.4-4. The THREAD Processor outputs are transmitted on the Pulse Request (PR) BUS.

Several control sequences are required to implement the flow of control. These relate to:

(1) Tracks that are killed directly during the pulse return processing:

- TIBDED Angle Group Deletion
- OT1 Ghost Detection
- OT5 Intercept Complete
- AOD Decoy Detection
- MG (IC) Intercept Complete



EVENT SEQUENCE

(1) Pulse request to assigned processor and memory
(2) Track accessed from database to assigned processor
(3a) Next pulse request and, (3b) completed status to RRA
(4) Update database

Figure 4.4-4. Data Flow for Pulse Return Processing

(2) Correlation:

- TIOT
- OBJCOR

(3) Discrimination

- POD, Passive
- AOD, Active

(4) Intercept Planning

- Intercept Plan Status Output
- Implementation Response

Tracks that are killed directly during pulse return processing (as contrasted with killed because of correlation or discrimination processing) are deleted by means of a marked track data base update sent from the THREAD Processor to the MEMORY Processor -- just as occurs during a normal data base update as shown in Figure 4.4-4. The MEMORY Processor either deletes the track entry or, if required, leaves the marked entry in the database for use by the URM.

Correlation processing is shown in Figure 4.4-5. The correlation functions are not as response-time sensitive as the pulse return processing functions. Thus, the objective of the approach is to minimize conflict with these higher priority functions. This is accomplished by two features of the approach: (1) Scheduling of correlation is by the RRA. Thus, the priority of the function is weighted against other system needs; and (2) The sifting of the track database for "close" tracks is done off-line. It would be appropriate for the RRA algorithm to age the request to allow time for this database sift.

A wide associative memory is a natural structure for the track database. With this structure the MEM Processor would have a very easy time marking the "close" tracks to be accessed.

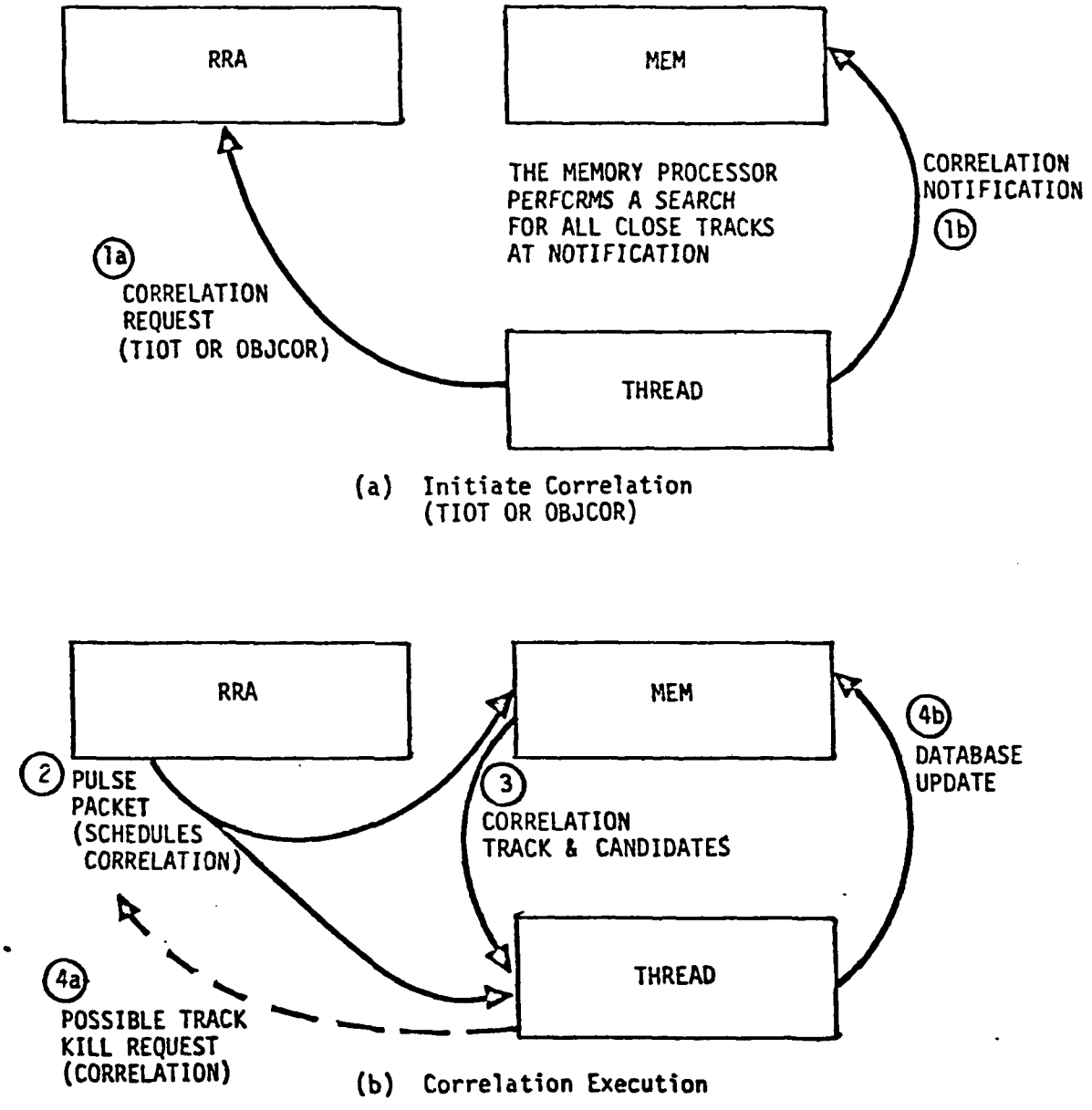


Figure 4.4-5. Correlation Processing

When a THREAD Processor is assigned for correlation, the MEM Processor simply transfers the track to be correlated, and the associated "close" tracks. The THREAD processor performs the correlation, updates the database and, if appropriate, request that the RRA kill the track.

A second approach is to have the Memory Processor perform the correlation. However, this would put a heavy additional computation load on the processor. We believe that a sleek special processor oriented solely toward database management will prove most cost effective in this area.

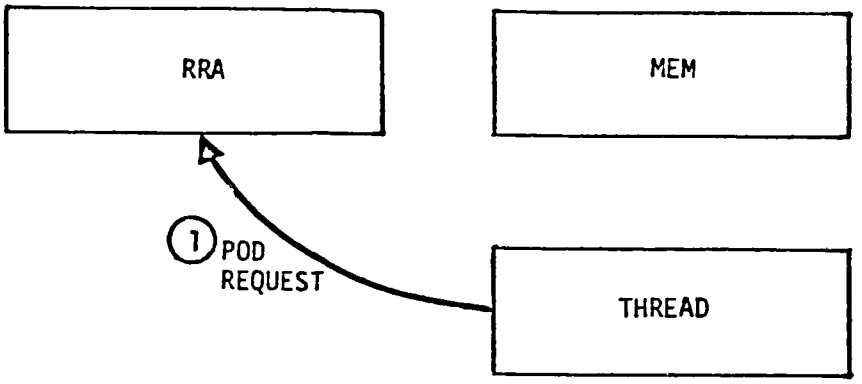
Passive discrimination normally (.975) results only in an update of the database that indicates that the object remains unclassified (see Figure 4.4-6). However, fragment identification would cause a kill track request to be issued to the RRA. Also, certain track characteristics cause the initiation of active discrimination.

Active object discrimination is a pulse return processing thread and is controlled identically to S/V, TI, and OT pulse return processing. This processing has a longer response time requirement. Thus, it has a lower scheduling priority.

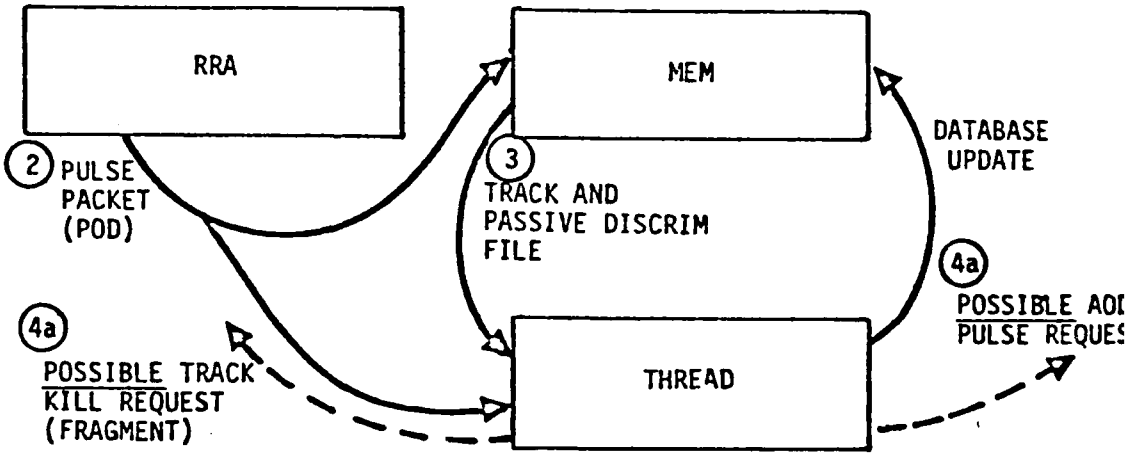
Intercept planning is initiated from AOD. This could either be initiated by a request to the RRA Processor or possibly be included within the AOD processing task.

4.4.6 Pulse Request (PR) BUS

The PR BUS transports data between the THREAD processors and the Memory Processor (MEM) for update of the database. This BUS also carries Pulse Request Packets to the SCHEDULER. The loading on the bus can be upper bounded by examining the maximum pulse rate of the radar. At 2000 pps the BUS loading would be:



(a) Initiation of Discrimination



(b) Passive Discrimination Execution

Figure 4.4-6. Discrimination Processing

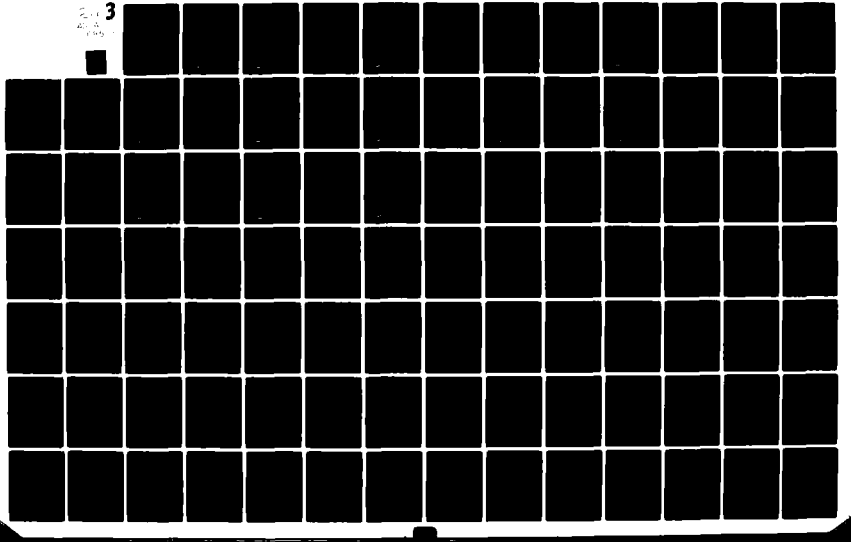
AD-A108 589 VERAC INC SAN DIEGO CA F/G 15/3.1

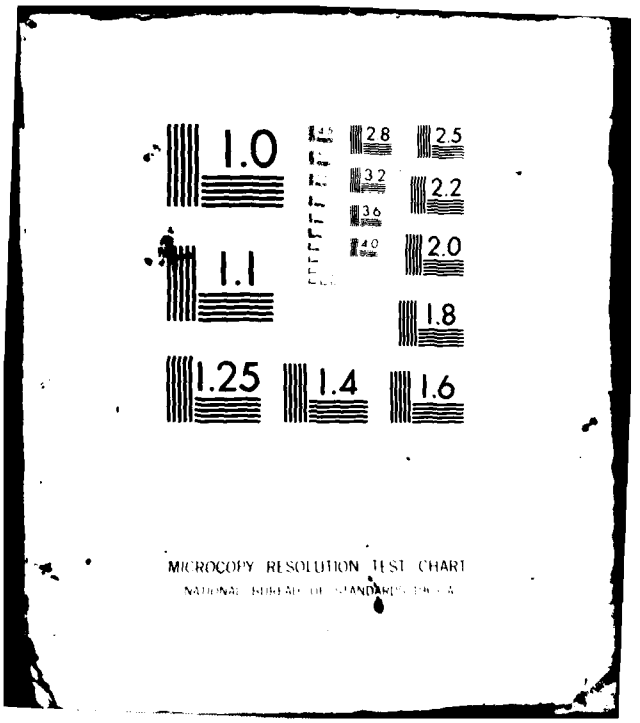
AN ANALYSIS OF MMCS NETWORK ARCHITECTURES TO SUPPORT THE DATA P--ETC(U)

DEC 80 J C TIERNAN DASG60-80-C-0017

UNCLASSIFIED R-008-80 NL

3




1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART

Pulse Request Packets (1600b/packet)	3.2 Mbps
Database Updates re pulses	9.6 Mbps
Other Database Updates (Correlation and Discrimination)	.48
TOTAL	13.08 Mbps

If the same bus architecture was used as with the RR BUS (1 Mwps/32b), capacity is available for a flexible demand access protocol and for a significantly higher radar pulse rate. This is within present technology limits.

4.4.7 SCHEDULING Processor

The Scheduling Processor takes pulse request packets and schedules pulses to the radar on a priority bases in accordance with pulse rate and energy constraints. The rate and energy constraints are set by the URM Processor and downloaded by means of the Resource Control (RC) BUS to the Scheduling Processor (see Figure 4.4-1).

The Scheduling Processor formats and sends pulse commands to the RADAR. It also sends, for each pulse requested, an ID Packet by means of the RC BUS to the RRA Processor.

The applications loading for the SCHEDULER is slightly less than 5 MIPS (Table 4.4-1). However, the functional requirement can be divided into MICRO and MACRO Scheduling stages in order to lessen the cost of the implementation. Thus, two serial 2.5 MIP processors appears to be an attractive implementation of this NODE.

4.4.8 Memory Processing/Database Structure

The architectural approach of Figure 1 includes a special purpose Memory Processor (MEM). This processor performs the following functions:

(1) Responds to Pulse Return Packets containing radar return information by fetching and placing the appropriate

Track/Group database entry on the RR BUS or by deleting killed entries.

(2) Responds to Database Updates Packets by updating the related entry or by deleting killed entries, if appropriate.

(3) Responds to Correlation Notices by searching the track database for objects that are close to the object to be correlated. Objects that are found to be "close" are tagged or noted in a list in preparation for retrieval.

(4) Responds to Pulse Return type packets that indicate correlation or discrimination by placing the track entry and related entries on the RR BUS.

(5) Responds to URM queries by placing requested track data on the RC BUS.

The memory processor is given functional responsibility only for processing that is intimately associated with the management of the central database.

There is a natural organization of the database memory into addressable words of 4800b/w.--the size of a single track or group entry. 11 or 12 bits of address would be required with this structure. The address might easily correspond to the track number. Content addressability is required for rapid determination of "closeness" for the correlation approach that we have advanced.

5.0 CONCLUSIONS

The Hybrid Architecture advanced here offers a number of singularly attractive characteristics:

(1) The approach allows the shifting of the functions addressed by a computer during a scenario so that the system loading remains balanced, and no processing capability is idle during peak loading.

This characteristic depends on the centralized pulse-return scheduling function located in the RRA, to retain the simplicity of design, and the centralized database structure.

(2) Fault tolerance is embedded in the parallel design. THREAD processors are envisioned as performing constant on-line testing. When a fault is reported by a processor to the RRA, this THREAD Processor is immediately descheduled. The problem of providing high processor reliability is then concentrated on the special-purpose RRA, MEM, and SCHEDULING processors.

This characteristic depends on the centralized scheduling and the completely redundant structure of the proposed THREAD processors.

(3) The computing speed of the THREAD processors need not be over 0.5 MIPS, except for end game processing, where 1.25 MIPS processors are required. Thus, the actual capability selected for these processors would be based on a trade-analysis of cost, redundancy, and operating system complexity. Thus, there is an opportunity to minimize cost.

(4) Maximum loading of the processor segments is determined by the radar pulse rate limit. Thus, simple protocols for the BUSSES, and MEM, RRA, and SCHEDULING processors can be implemented.

The architecture requires a 1 Mwps at 32b/w BUS on the input and an identical BUS on the output. Given 10 accesses/pulse-return as a bound, 20,000 access/sec (50 μ-sec/access) are required to support a 2000 pps radar rate. These data and access rates are easily obtained with present technology.

(5) Both the RRA and SCHEDULING processors are special-purpose processors with control structures dedicatd to performing the required algorithms. These processors, and the MEM processor, would be designed with emphasis on fault tolerance. The RRA and SCHEDULING processors both are implemented as two-stage processors. The URM might also be a special purpose processor.

The architecture can be contrasted with the Thread Processing Architecture or the Centralized Architecture that were also examined in this study.

The Thread Architecture has a limited number of functions assigned to each computing NODE. This requires that redundancy be added to the NODE structure in order to achieve high reliability. The redundancy can be provided for a NODE either internally to each processor, or by providing multiple processors. Also, distinct hardware and software components may be required for different NODES, increasing the cost and complexity of design and development. A Thread Architecture has the characteristic that substantial application processing capability is not being used at the maximum loading point in the worst-case scenario.

The Centralized Architecture has the major fault of not taking advantage of the response-time/throughput loading disparity characteristic of the Site Defense Problem. The processing power must support the peak throughput loading. A very powerful and costly processor must be provided to meet this load. Managing the central processor requires a significant operating system overhead in addition.

VERAC has analyzed the loading and throughput requirements for the BMD Site Defense Processor. This analysis has motivated the development of an approach to the processing architecture that is characterized by parallel processors scheduled in real-time to perform needed functions. The approach also depends on a central, track database managed by a dedicated processor, and on a single scheduling and control point.

REFERENCES

[1] "Computer Program Configuration Item Preliminary Part II Specification Engagement Software, Volume II, Books 1, 2, and 3," TRW CI 11982-10751019, dated 22 October 1973.

[2] "BMD System Technology Program Product Specification for Computer Program Engagement Software, CR3, Volume II, Books 1, 2, 3, 4, and 6," TRW CI 17773-10751019B, dated 1 July 1973.

[3] "Advanced Data Processing Subsystem Investigation, Mini/Micro Computer Systems Evaluation Results," Volume XVII, MDAC Document No. MDC G8418, 7 December 1979.

[4] "April, 1978 Process Design Snapshot Report for Cycle 6 TAP," TRW.

Appendix A

Detailed Process Description

This Appendix provides a functional representation of a portion of the BMD terminal defense data processing. The portion represented includes radar assimilation and scheduling, S/V return processing, angle group tracking (track initiation), object tracking and discrimination, intercept planning and interceptor control.

The functional representation consists of a set of PRIMITIVES, each of which is intended to represent a logically cohesive unit of processing. Each PRIMITIVE indicates processing inputs, outputs, processing load (CDC machine language instructions or MLI), significant accesses to data files, and the evolution of instances at the output of each PRIMITIVE.

This process description was derived from available documents, from extensive technical discussions with McDonnel-Douglas personnel (particularly Dr. Shiang Liu), and, where necessary to fill gaps, from estimates based upon a general understanding of the problem environment. In particular, the sources are as follows (and are referenced to the associated number in the remainder of this appendix):

References for data items

[1] "Computer Program Configuration Item Preliminary Part II Specification Engagement Software, Volume II, Books 1, 2, and 3," TRW CI 11982-10751019, dated 22 October 1973.

[2] "BMD System Technology Program Product Specification for Computer Program Engagement Software, CR3, Volume II, Books 1, 2, 3, 4 and 6," TRW CI 17773-10751019B, dated 1 July 1978.

[3] "Advanced Data Processing Subsystem Investigation , Mini/Micro Computer Systems Evaluation Results," Volume XVII, MDAC Document No. MDC G8418, 7 December 1979.

[4] "April, 1978 Process Design Snapshot Report for Cycle 6 TAP," TRW.

[5] Private communication with S. Liu, McDonnel-Douglas.

[6] Estimate

A-1 Processing Loads (Timing Models)

Available software documentation ([1] and [2]) provides timing models for much of the terminal defense data processing. These models were obtained by linear regression applied to individual processing TASKs. Results are expressed in μsec as executed on a CDC7700. For purposes here, the resulting equations are more detailed than necessary. We obtained simplified forms by eliminating relatively insignificant contributions and otherwise focusing on principal effects. Generally, this left us with a model involving a dependence upon a few scenario-dependent parameters.

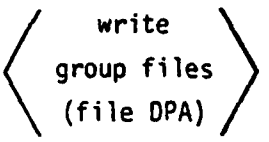
In the individual PRIMITIVE descriptions, these simplified timing models are given. When scenario-dependent parameters are involved, ranges and nominal values are provided to arrive at specific CDC7700 MLI processing loads.

As available timing models pertain to entire TASKs, whereas several PRIMITIVES were generally identified for each TASK, it was necessary to estimate the separate processing loads for individual PRIMITIVES by partitioning the TASK total.

The Radar Returns Assimilation and Radar Scheduling timing models were supplied to us by McDonnel-Douglas [5], as the available linear regression model [1] was not useful to us. In the case of the scheduling function, we divided the processing load associated with Macroscheduling and Microscheduling evenly (based upon existing results which indicate a nearly equal throughput requirement for these function [3]).

A-2 Data Access and Transfer

Two mechanisms are used in the PRIMITIVES to represent data access and transfer as illustrated in figure A-1. The "direct" mechanism is associated with input and output queues. The associated data transfers are then on the communication links between PRIMITIVES. The "indirect" mechanism is illustrated in figure A-1 by



Here, the channel for movement of data is not explicitly modeled. Rather the processing delays and contention for resource are modeled by reference to an external resource or device.

Generally, we associate to the output queues all data which are needed immediately by a subsequent PRIMITIVE. Corresponding inputs have associated the same data. Other data accesses and transfers, for example, to and from an object track file, are represented by the "indirect" mechanism.

Note that the "direct" transfers are not necessarily implemented in a particular architecture by a direct communication link. In the existing CDC7700 architecture, for example, essentially all data transfer are through the LCM (large core memory).

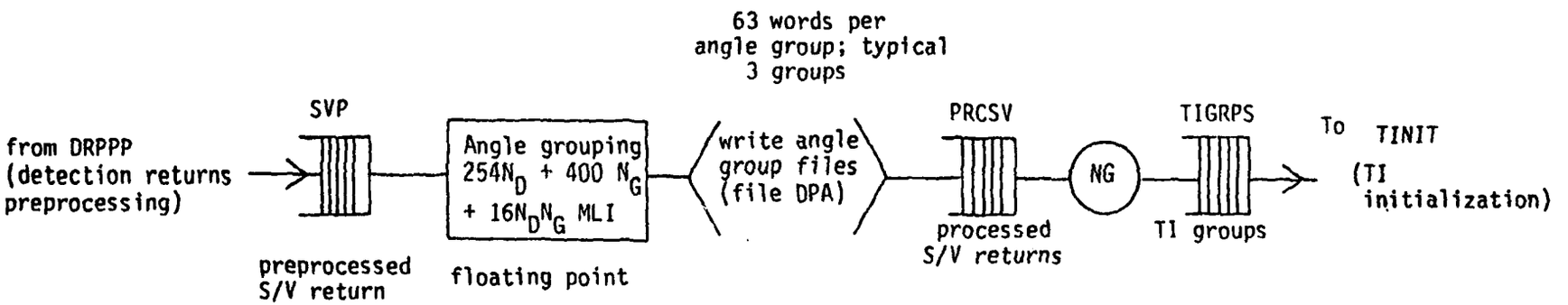
A-3 Process Evolution Principles

As presently constructed, PRIMITIVES involve a number of scenario-dependent parameters related to the evolution of the status of objects (and interceptors) as they are detected, tracked, classified and intercept. These parameters appear in the final splitting to obtain outputs in each PRIMITIVE.

Primitive: ANGRP

Angle group processing

Detection returns processing function



Nominal parameters

N_D: 10 (5-15)

N_G: 3 (2-4)

Implies 4220 MLI

Figure A-1 PRIMITIVE ANGRP, illustrating two mechanisms for representing data access and transfer

As PAES does not maintain the identity of individual tracks, a flow-oriented representation of the process evolution is necessary. We have adopted a finite-state flow model. The process states are indicated in Table 1. Objects begin in the state "SV", i.e., have associated search/verify returns, and evolve through states TI → OT1 → OT2 → OT3 → OT4 → OT5 → complete, as illustrated in figure A-2. Some objects are dropped at an intermediate stage. Further, entering OT3 is associated with the generation of active discrimination pulses (AOD), entering OT4 is associated with intercept planning and OT5, with interception (MG).

These Process states have been identified to correspond to the principal functional decomposition of the process. In so doing, approximations have been introduced. For example, in the interval following object flight through the commit contour but prior to interceptor launch, some objects have been classified as RVs, but others have not. As the RV class is substantially larger than the set of threatening but unclassified objects, we have chosen to ignore the latter, effectively assuming that active discrimination (successfully) classifies all objects before they reach the commit contour. Additional refinement is, of course, always possible.

In figure A-2, the mean dwell times for each state are indicated, e.g., .25 sec for state "TI". The sum of the mean times from initial detection to completion of the intercept is seen to be 10 seconds.

Implementation of the process state model involves fractionally splitting the flow at certain points in the process, typically, at the output of a track filter primitive, or at the output of a discrimination primitive. The parameters of the splitting are scenario-dependent and are chosen according to two principles:

(1) to achieve the indicated mean dwell times in each process state, and

(2) to achieve desired fractions of target and object types.

In the next section, we obtain general model relations to be used in computing scenario-dependent splitting parameters.

Table A-1 Process Evolution States

SV	search verify processing
TI	angle group tracking (track initiation)
OT1	object tracking before any object discrimination
OT2	object tracking with passive object discrimination
OT3	object tracking with active object discrimination
OT4	object tracking during intercept planning
OT5	object tracking during intercept
POD	passive object discrimination for unclassified objects (objects in state OT2)
AD	active object discrimination for unclassified objects (objects in state OT3)
MG	missile guidance

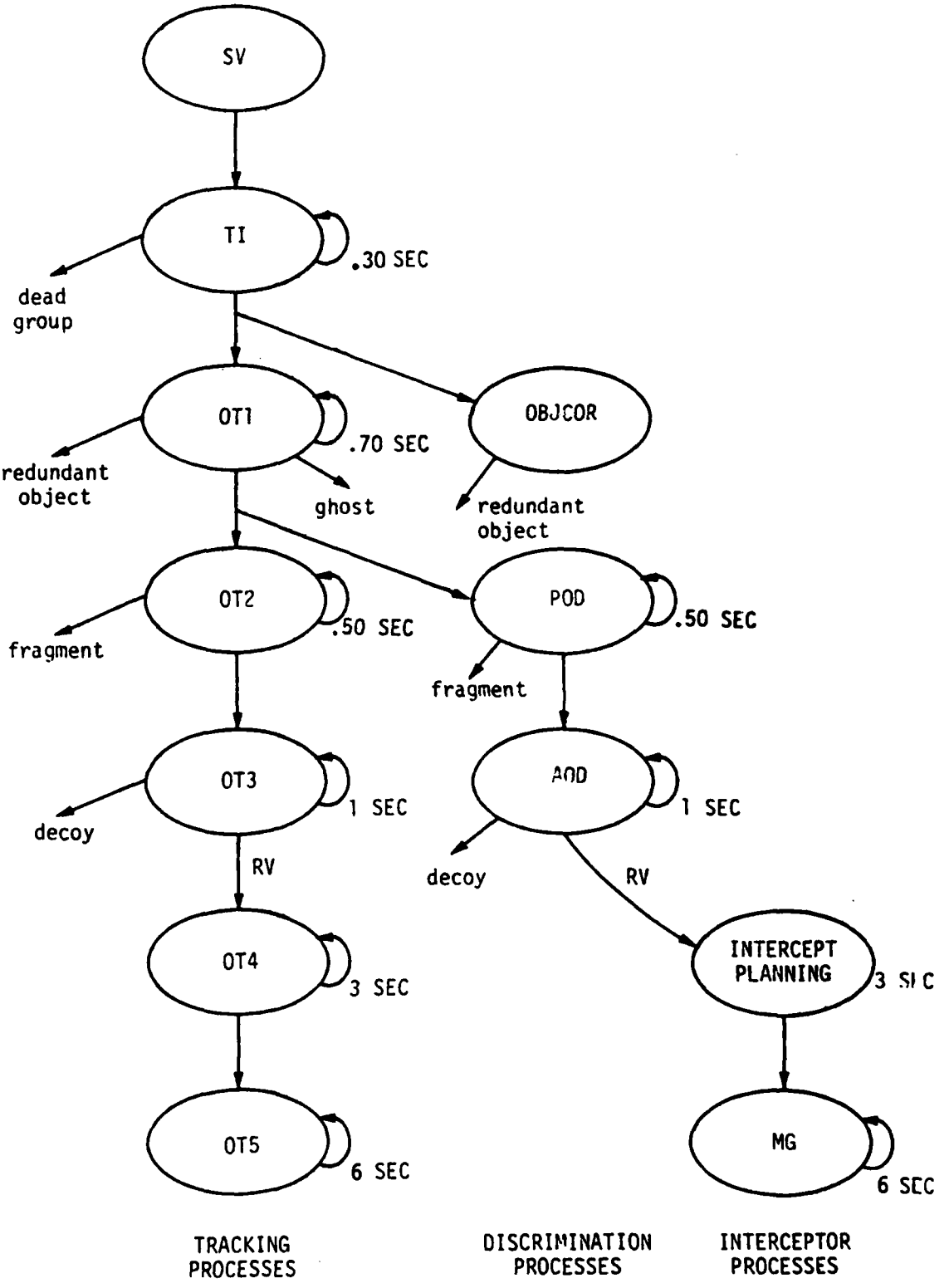
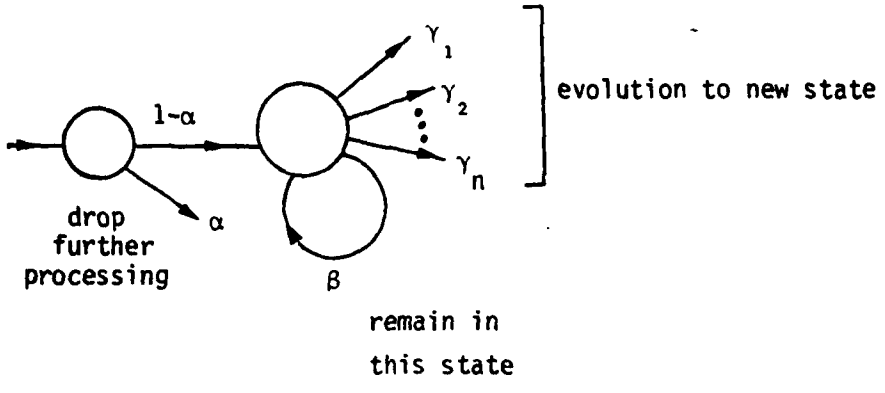


Figure A-2. Process Evolution

Model Relations

The typical element of process evolution represented within a PRIMITIVE is represented graphically as follows:



The first split in the processing evolution generally corresponds dropping further processing (e.g., dropping redundant track). The second splitting in the process evolution generally corresponds to partial evolution to one or more new states. A fraction β of the instances continue in the same state, while various fractions, γ_1, ..., γ_n, evolve to new states. The mean dwell time, or number of work-unit cycles through this PRIMITIVE for each work unit introduced, for this exponential model is given by

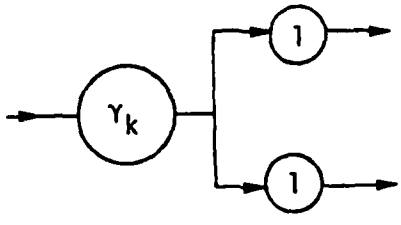
$$\tau = [1 - (1 - \alpha)\beta] \sum_{n=1}^{\infty} n[1 - (1 - \alpha)\beta]^n$$

$$= \frac{1}{1 - (1-\alpha)\beta} \qquad \text{(A-1)}$$

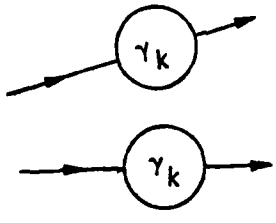
It is assumed that each instance either remains in this state or evolves to a distinct new state, so that

$$\sum_{i=1}^{n} \gamma_i = 1 - \beta \qquad \text{(A-2)}$$

Note that evolution to a new state may generate multiple events so that one might have, for example



which is more directly represented as a replication:



The assumed scenario provides "ultimate" fractions for each object type. To obtain "single cycle" fractions, one must account for the fact than an instance passes through the process repeatedly.

The ultimate fraction, u_0, corresponding to the single pass fraction, α, is given by

$$\mu_0 = \alpha \underbrace{\left[1}_{\substack{\text{first}\\\text{pass}}} + \underbrace{\beta\,(1-\alpha)}_{\substack{\text{second}\\\text{pass}}} + \underbrace{\beta^2\,(1-\alpha)^2}_{\substack{\text{third}\\\text{pass}}} + \cdots\right] = \frac{\alpha}{1-\beta(1-\alpha)} \qquad \text{(A-3)}$$

To solve (A-1), (A-3) it is convenient to let

$$\beta' = \beta(1-\alpha).$$

Then (A-1), (A-3), respectively, become

$$\tau = \frac{1}{1-\beta'}$$

$$\mu_0 = \frac{\alpha}{1-\beta'}.$$

These are easily solved to yield

$$\beta' = 1 - \frac{1}{\tau}$$

$$\alpha = \mu_0\,(1-\beta')$$

$$\beta = \beta'/(1-\alpha).$$

If $\mu_1, \ldots, \mu_N$ represent the ultimate fractions of states evolving from the present state, so that

$$\mu_0 + \sum_{i=1}^{N} \mu_i = 1, \qquad \text{(A-4)}$$

it follows from (A-2) and (A-4) that

$$\gamma_i = \kappa \eta_i , \quad i = 1, \ldots, N.$$

where

$$\kappa = (1 - \beta)/(1 - \mu_0) = 1 - \beta'.$$

Certain events are initiated once when a new state is entered. The fraction associated with these events is also κ, since the ultimate fraction, a ratio to initiations of this state, is unity.

A-5 Process Evolution Model

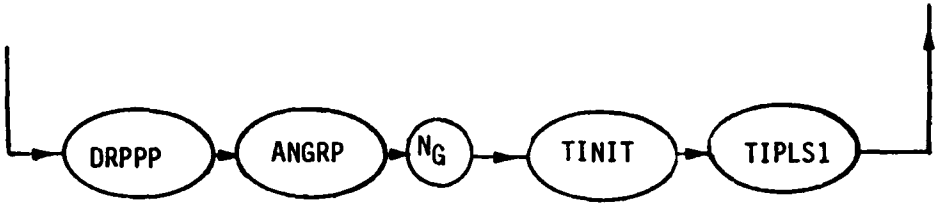
In the following, we have extracted those (sets of) PRIMITIVES with scenario-dependent splitting parameters. Each parameter is identified, and numerical values computed for them. These computations are based upon the model relations presented in the previous section and the scenario defined in Table A-2. The process assumptions are further defined in Table A-3.

Table A-2. Assumed Scenario

	OT1			OT2			OT3		
	No.	μ	α or γ	No.	μ	α or γ	No.	μ	α or γ
redundants	8	.0571	.004						
ghosts	10	.0714	.005						
fragments	20			20	.164	.016			
decoys	70	.8714	.062	70	.836	.085	70	.686	.035
RVs	32			32			32	.314	.016
Total Objects	140			122			102		

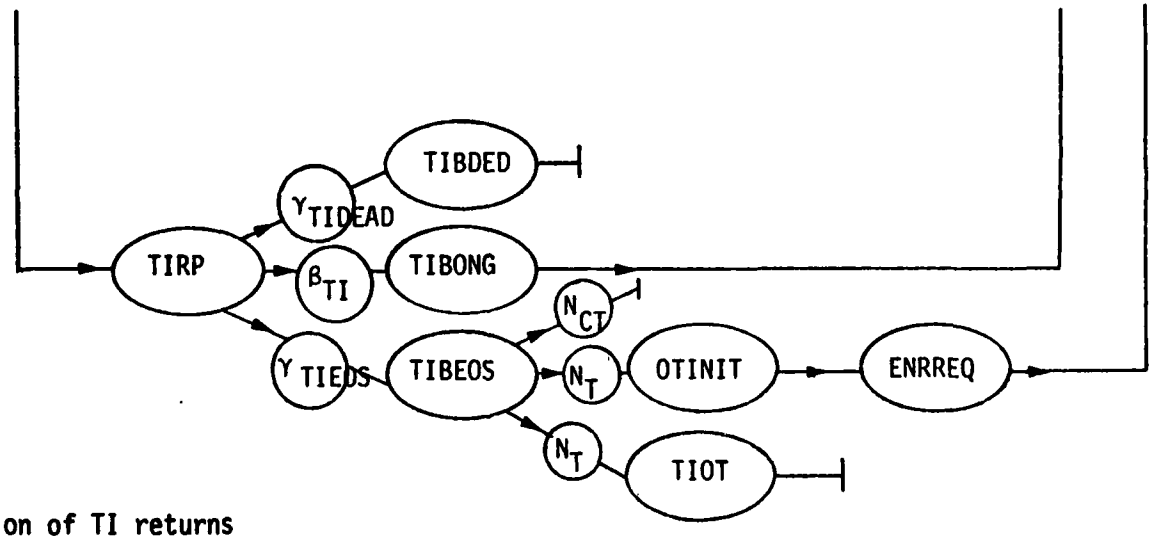
Table A-3. Mean Times In Each State

State	SV	TI	OT1	OT2	OT3	OT4	OT5	AD	MG
mean time (ms)	100	300	700	500	1000	3000	6000	1000	6000
associated cycle time	100	50	50	50	50	50	25	250	20
# τ cycles		6	14	10	20	60	240	4	300
β'		.833	.929	.900	.950	.983	.996	.750	.997
β		.833	.933	.915	.983	.983	.996	.750	.997
κ		.167	.071	.085	.050	.017	.004	.250	.003



N_G: # of angle groups created from a S/V return (3)

SV EVOLUTION



γ_{TIDEAD} (.008): fraction of TI returns which die (no targets)

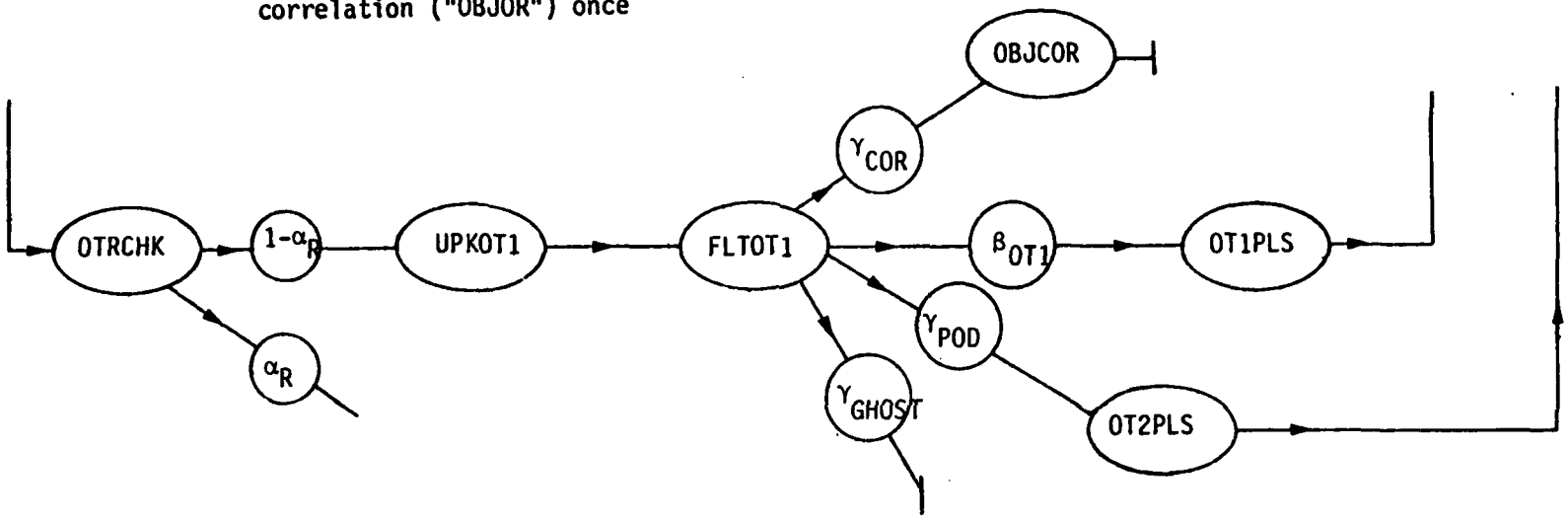
β_{TI} (.833): fraction of TI returns which continue (models dwell time)

γ_{TIEOS} (.159): fraction of TI returns which are end of sequence

N_{CT} (1): number of cross targets in an EOS angle group

N_T (2): number of targets to be tracked in an EOS angle group

TI EVOLUTION



α_R (.004): fraction of OT1 tracks dropped as redundant

β_{OT1} (.933): fraction of OT1 tracks which continue as OT1 (models dwell time)

γ_{COR} (.062): fraction of OT1 tracks which are submitted for OBJCOR test = $\kappa(1-\mu_{REDundant}-\mu_{GHost})$

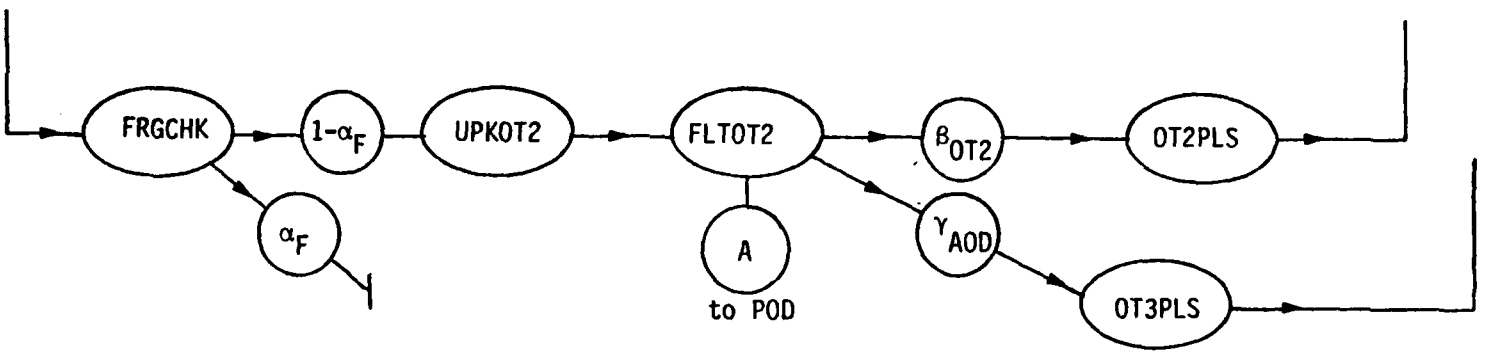
γ_{POD} (.062): fraction of OT1 tracks which are submitted for passive object discrimination (and thus transition to state OT2) = $\kappa(1-\mu_{GHost} = \mu_{GHost})$

γ_{GHOST} (.005): fraction Of OT1 tracks which are found to be ghost = $\kappa\mu_{GHost}$

OT1 EVOLUTION

VERAC Incorporated

In this state, each return is also subjected to passive object discrimination (by "POD").

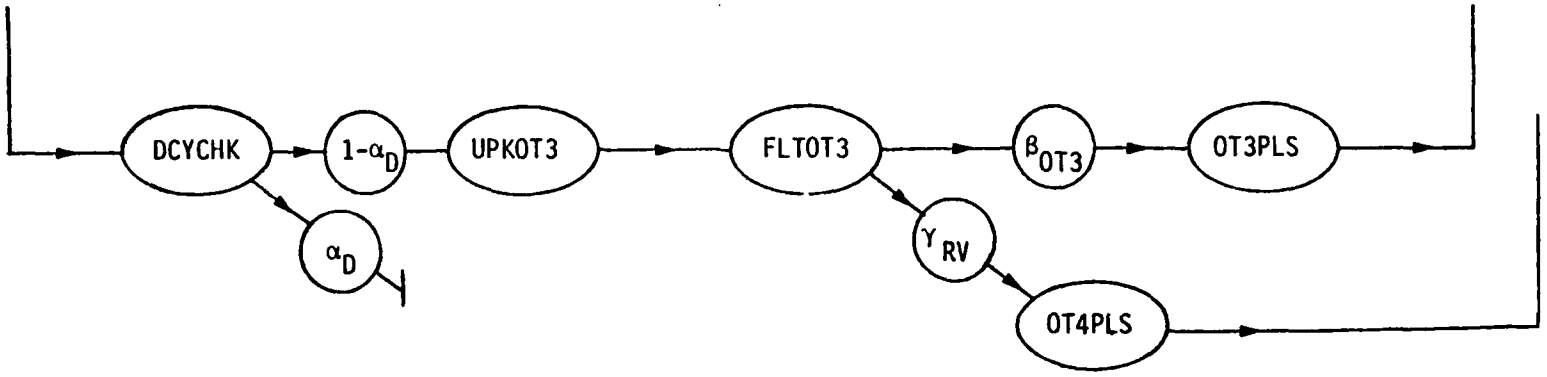


α_F : (.016) fraction of OT2 tracks dropped as fragments

β_{OT2}: (.915) fraction of OT2 tracks which continue as OT2 (and therefore activates POD)

γ_{AOD}: (.085) fraction of OT2 tracks (decoys and RVs) which enter active discrimination (and therefore transition to state OT3)

OT2 EVOLUTION



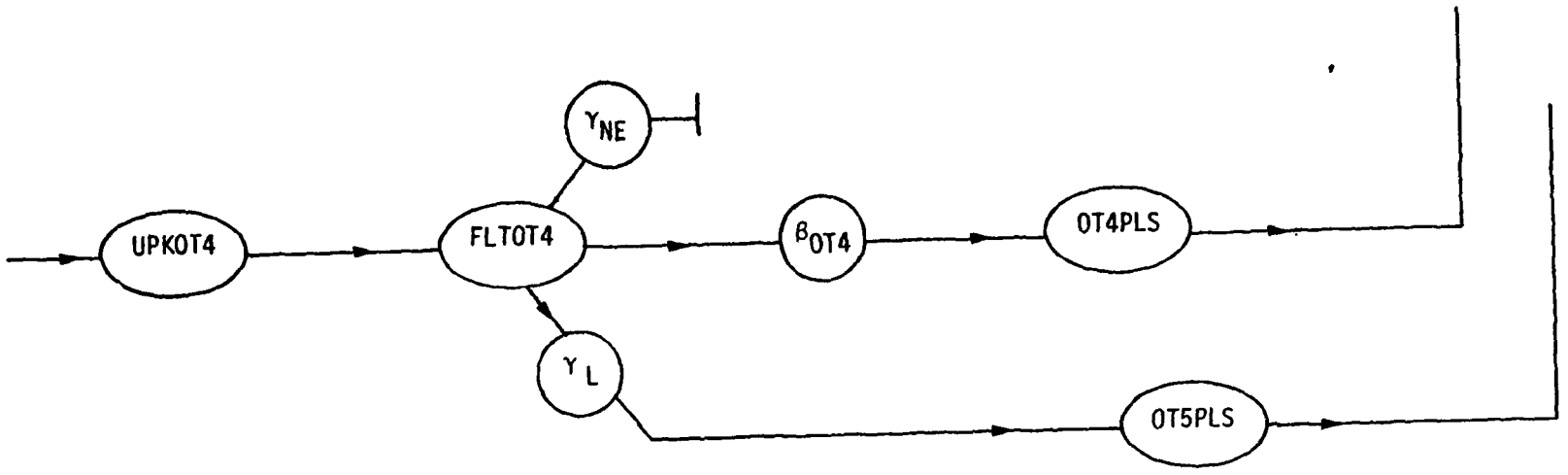
α_D: fraction of OT3 tracks dropped as decoys
(.035)

β_{OT3}: fraction of OT3 tracks which continue as OT3
(.984)

γ_{RV}: fraction of OT3 tracks which are classified as RVs (and therefore transition to state OT4)
(.016)

Note: It is assumed that active discrimination successfully classifies all objects

OT3 EVOLUTION



β_{OT4}: fraction of OT4 tracks which continue (prelaunch object tracking still underway)
(.983)

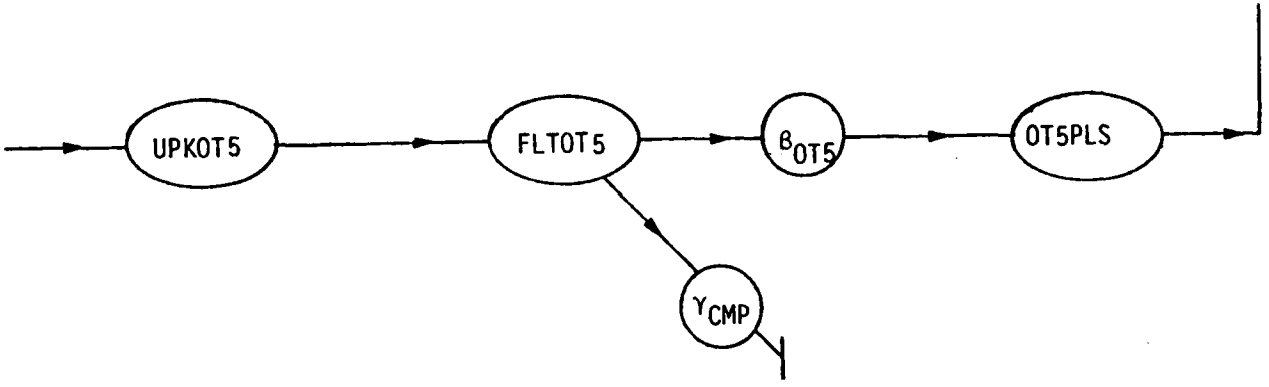
γ_L: fraction of OT4 tracks associated with the launch of an interceptor (and therefore transition to state OT5)

(.008) = $(1 - \beta_{OT4})\ f_{int}$, where f_{int} is the fraction of OT4 tracks which are intercepted

γ_{NE}: fraction of OT4 return processing terminated because OT4 track is not engaged

(.009) = $(1 - \beta_{OT4})\ (1 - f_{int})$

OT4 EVOLUTION

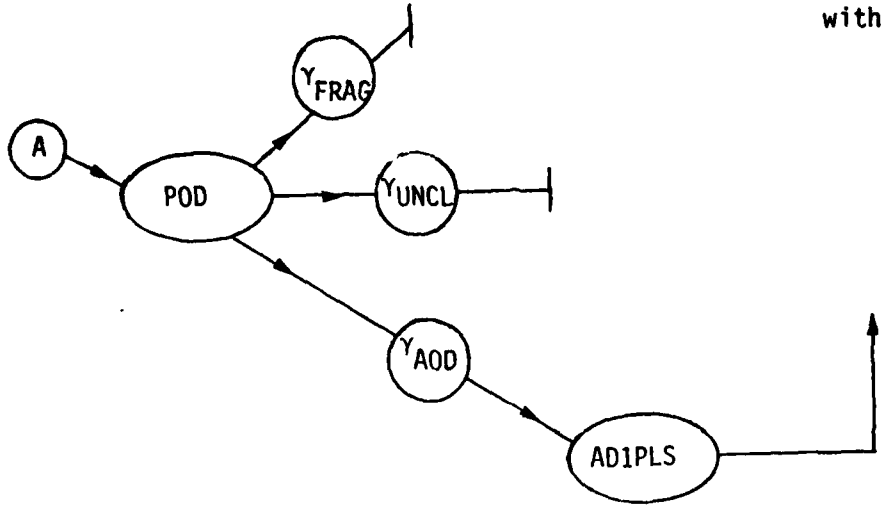


β_{OT5}: fraction of OT5 tracks which continue (intercept still underway)
(.996)

γ_{CMP}: fraction of OT5 tracks which stop (intercept complete)
(.004)

OT5 EVOLUTION

POD evolution is coordinated with state OT2 evolution

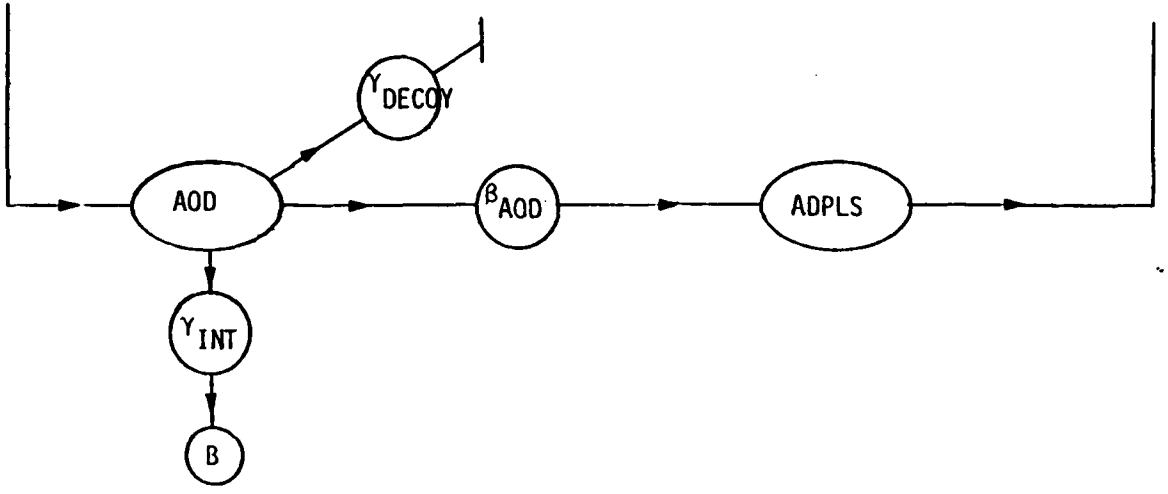


γ_{FRAG}: fraction of objects classified as a fragment
(.016) $= \beta_{OT2}\,\alpha_F$

γ_{UNCL}: fraction of objects remaining unclassified and not yet submitted to active discrimination
(.900) $= \beta_{OT2}(1 - \alpha_f)$

γ_{AOD}: fraction of objects submitted to active discrimination
(.084)

POD EVOLUTION

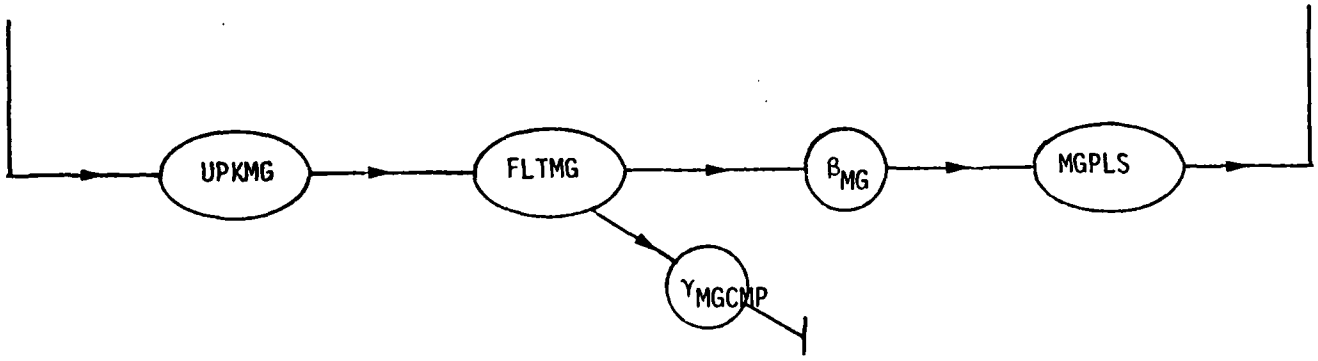


β_{AOD} (.750): fraction of objects which continue undergoing active discrimination

γ_{DECOY} (.172): fraction of objects undergoing active discrimination which are classified as decoys

γ_{INT} (.078): fraction of objects undergoing active discrimination classified as RVS, therefore initiating intercept planning and terminating active discrimination

AD EVOLUTION



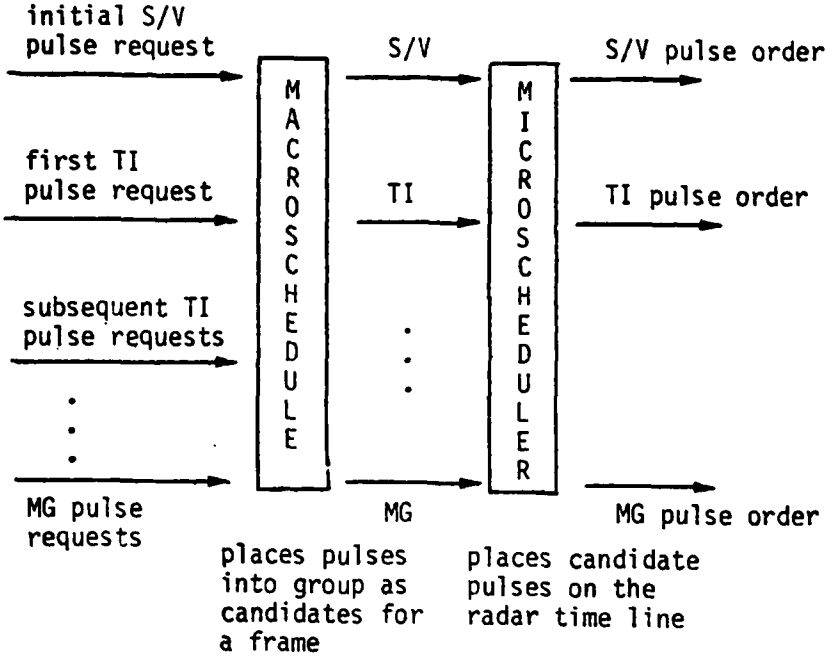
β_{MG}: fraction of interceptors continuing
(.997)

γ_{MGCMP}: fraction of intercepts completed
(.003)

MG EVOLUTION

Radar Scheduling Model

Radar scheduling is grossly decomposed into MACROSCHEDULER and a MICROSCHEDULER. While tne several radar pulse types are handled in a single data structure (in the present implementation), it is necessary to explicitly distinguisn each type in the functional model.



The macroscheduler prepares each pulse type by processing specific to each type, then creates groups of candidate pulses suitable for detailed (micro-) scheduling within a frame. Generally, pulse requests include a requested transmission time sometime after the time of receipt at the macroscheduler. Thus, the macroscheduler model must incorporate a representation of these time differences. Further, the macroscheduler resolves the radar rate constraint, a further effect to be modeled.

The macroscheduling of each pulse type is modeled by three PRIMITIVE functions:

radar pulse parameter computations

models deferral to nominal transmission time and framing delay

models radar rate constraint

Only the first PRIMITIVE involves a processing load. The latter two PRIMITIVES are used to model delays, as indicated above.

Deferral to Nominal Transmission Time

Pulses are given a nominal transmission time by the TASK which determines that a new pulse is required. This time is such as to achieve a fixed, interpulse interval in order to achieve sufficient tracking accuracy. In this flow-oriented model, pulses do not have an individual identity so that an effective delay to represent the deferral to the nominal transmission time must be introduced. Initially, a delay specific to each process state will be provided for. Appropriate values for the delays will be arrived at by trial-and-error through use of PAES program.

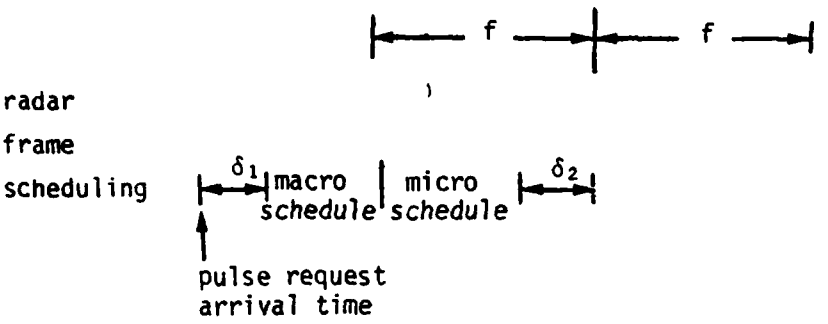
(In checking out the process description, where no architecture effects are present, the entire delay -- except for the Δt integration steps through non-zero delay primitives -- can be located in this model for deferral to nominal transmission time. The selected values would then be taken to be the desired port-to-port response times less the fixed delays incurred in other primitives on the relevant thread.)

Framing Delay

The use of a radar scheduling frame was introduced in the existing terminal defense data processing to achieve efficiency in (micro-)

scheduling and to meet TASK scheduling requirements in the CDC 7700 implementation. New scheduling concepts are probably appropriate for a distributed data processing implementation.

The use of a frame introduces a delay, as suggested in the illustration below:



The total time from pulse request arrival to the beginning of the frame into which the pulse is scheduled is given by

$$\delta_1 + \tau_{macro} + \tau_{micro} + \delta_2$$

as indicated in the illustration. The processing times τ_{macro} and τ_{micro} are explicitly modeled in distinct PRIMITIVES. The delay δ_1 is represented in the operating system. Delay δ_2 is incurred due to the discretization of the radar time line into frames. On average, this delay is f/2. Pulse requests are distributed over the frame, of length f, so that, on the average (and not providing priority to any particular pulse types) a further delay of f/2 is incurred. Thus the total framing delay is, on average, f.

Combined Model for Deferral to Nominal Transmission Time and Framing Delay

The total delay associated with the deferral to nominal transmission time and the framing delay is taken to be, simply,

$$\delta_{NT} + f$$

with δ_{NT} to be adjusted, by examination of simulation results, to achieve the effective desired interpulse times when this is possible (i.e., $\delta_{NT} > 0$). A heavy scenario can lead to the choice $\delta_{NT}=0$ and yet not achieve desired port-to-port response times. In these cases, the presence of the framing delay represents a realistic contribution to the actual port-to-port response time.

Radar Rate Constraint Model

The macroscheduler accounts for the radar rate constraint (R pps). This is modeled by a coupling between the primitives RDCXX for the various pulse types.

In a time increment Δt, $R\Delta t$ pulses can be handled. If

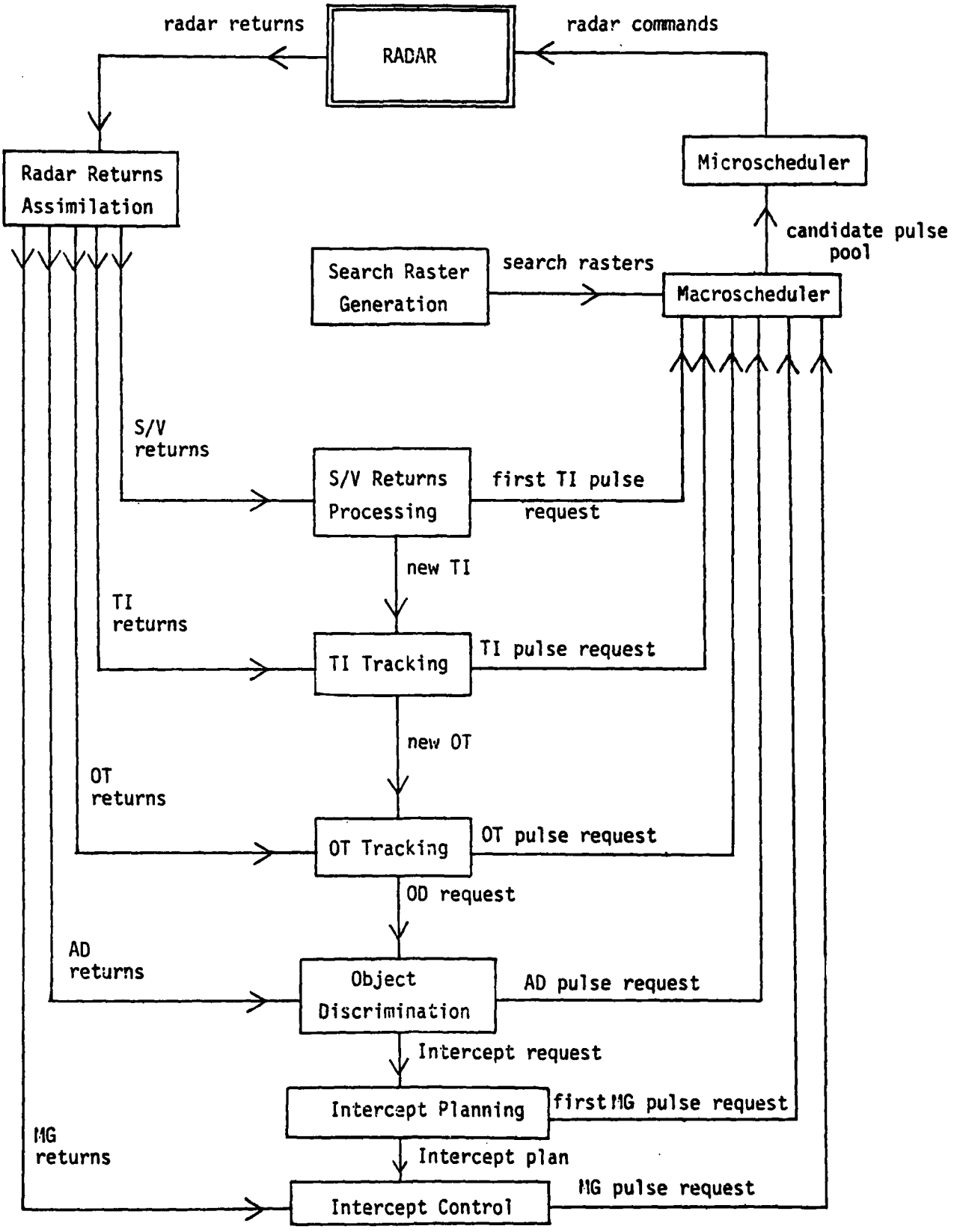
$$\sum \text{input queue lengths for each pulse type} \leq R\Delta t$$

the constraint is not active and each primitive is a zero-delay primitive. Otherwise, the total rate must be limited. While a priority scheme might be used, we take here a proportional model. That is, we take

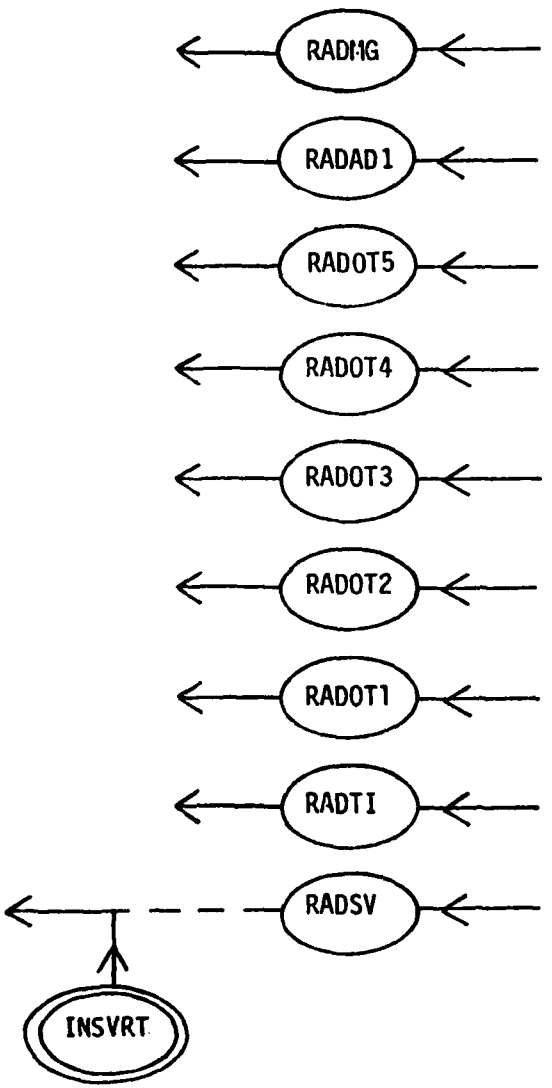
$$\alpha = \frac{R\Delta t}{\text{input queue lengths for each pulse type}}$$

and take as rates for the respective pulse types:

$$\alpha \times \text{queue length(pulse type)}/\Delta t$$



PROCESS OVERVIEW

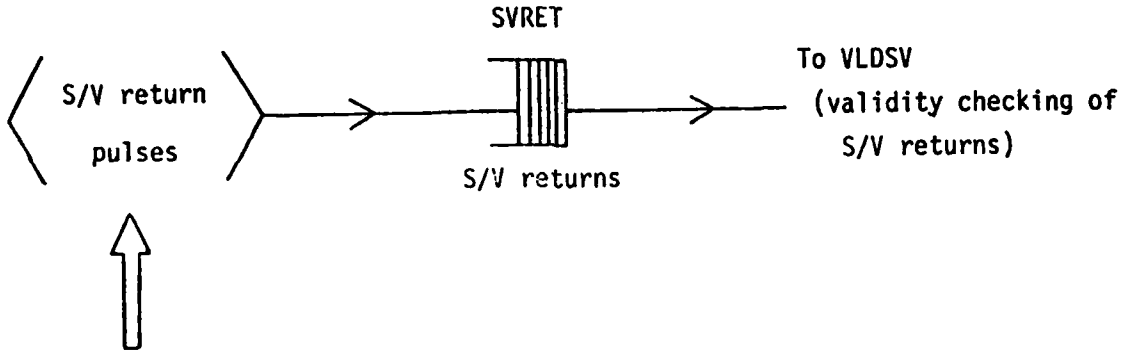


RADAR RETURNS MODEL

Primitive: INSVRT

Input S/V returns model

zero delay



scenario input data from file

The principal input driving the process is the S/V returns history.

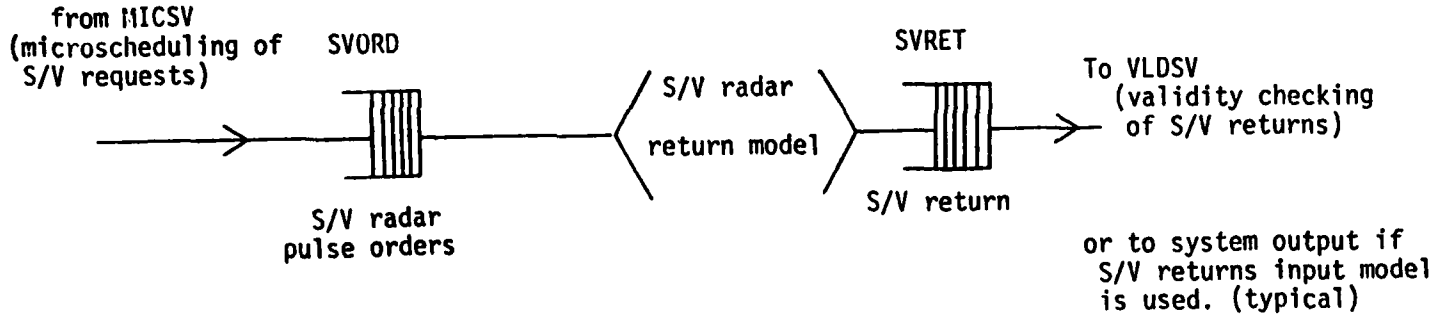
[for testing, use analytic model with uniform rate 10 pps]

Primitive: RADSV

S/V radar return model

Radar model

May be zero delay (if round trip travel time is "small")



Radar model is a pipeline model:

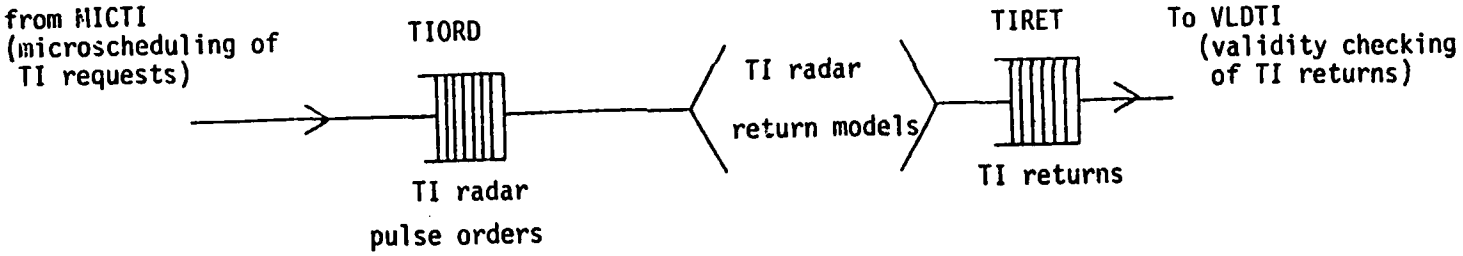
$$\text{rate} = \frac{\text{\# in SVORD queue}}{\text{round trip travel time}}$$

use default resource model (with zero delay)

Primitive: RADTI

TI radar return model

May be zero delay (if round trip travel time "small")



Return model is a pipeline model:

$$\text{rate} = \frac{\text{\# in TIORD queue}}{\text{round trip travel time}}$$

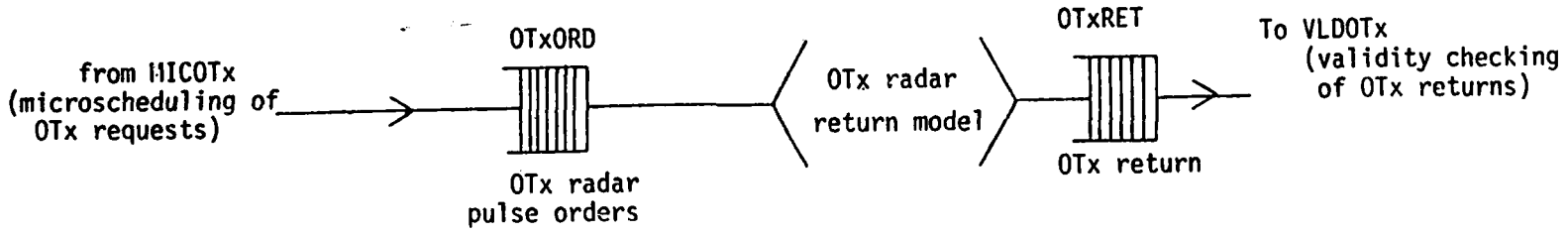
use default resource model (with zero delay)

Primitive: RADOTx (x = 1,2,3,4,5)

OTx radar return model

Radar model

May be zero delay (if round trip travel time is "samll")



Model:

$$\text{rate: } \frac{\text{\# in OTxORD queue}}{\text{round trip travel time}}$$

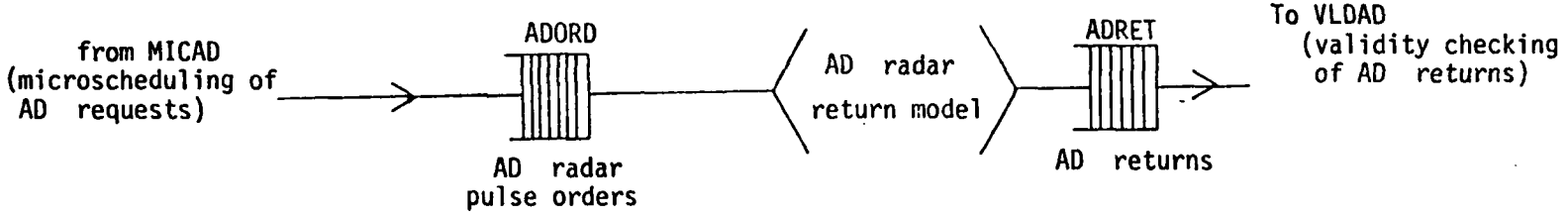
use default resource model (with zero delay

Primitive: RADAD

AD radar return model

Radar model

May be zero delay (if round trip travel time is "small")



Model:

$$\text{rate} = \frac{\text{\# in ADORD queue}}{\text{round trip travel time}}$$

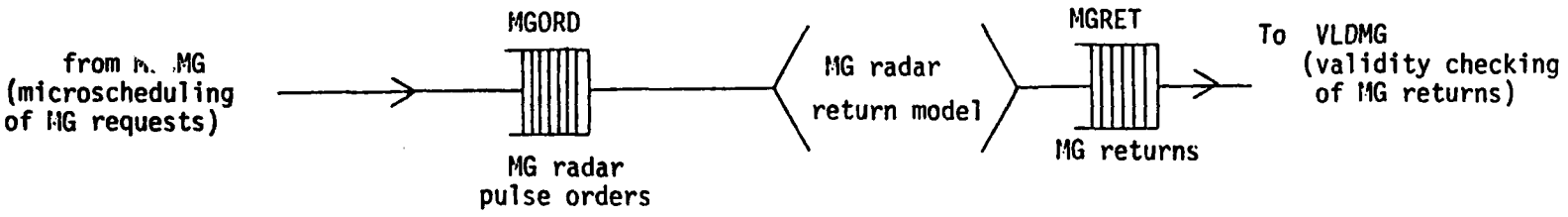
delay = 150 ms (Models 'D1' type AD pulse sequences)

Primitive: RADMG

MG radar return model

Radar model

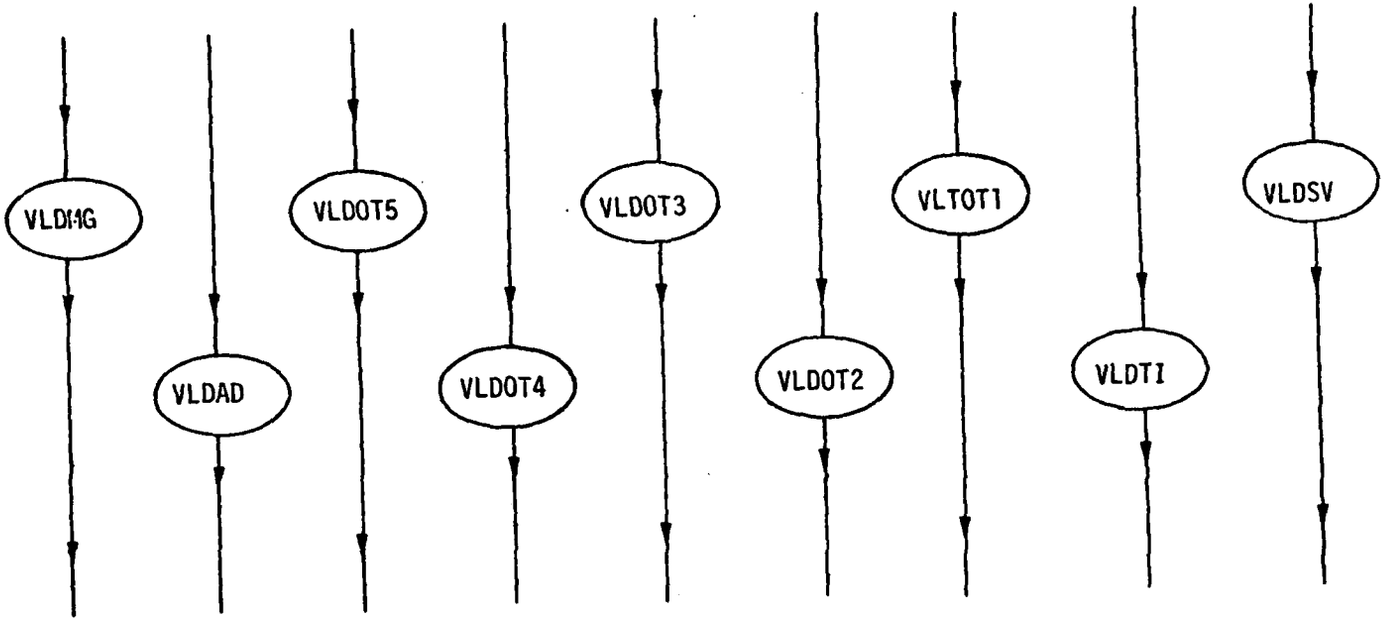
May be zero delay (if round trip travel time is "small")



Model:

$$\text{rate} = \frac{\text{\# in MGORD queue}}{\text{round trip travel time}}$$

use default resource model (with zero delay)



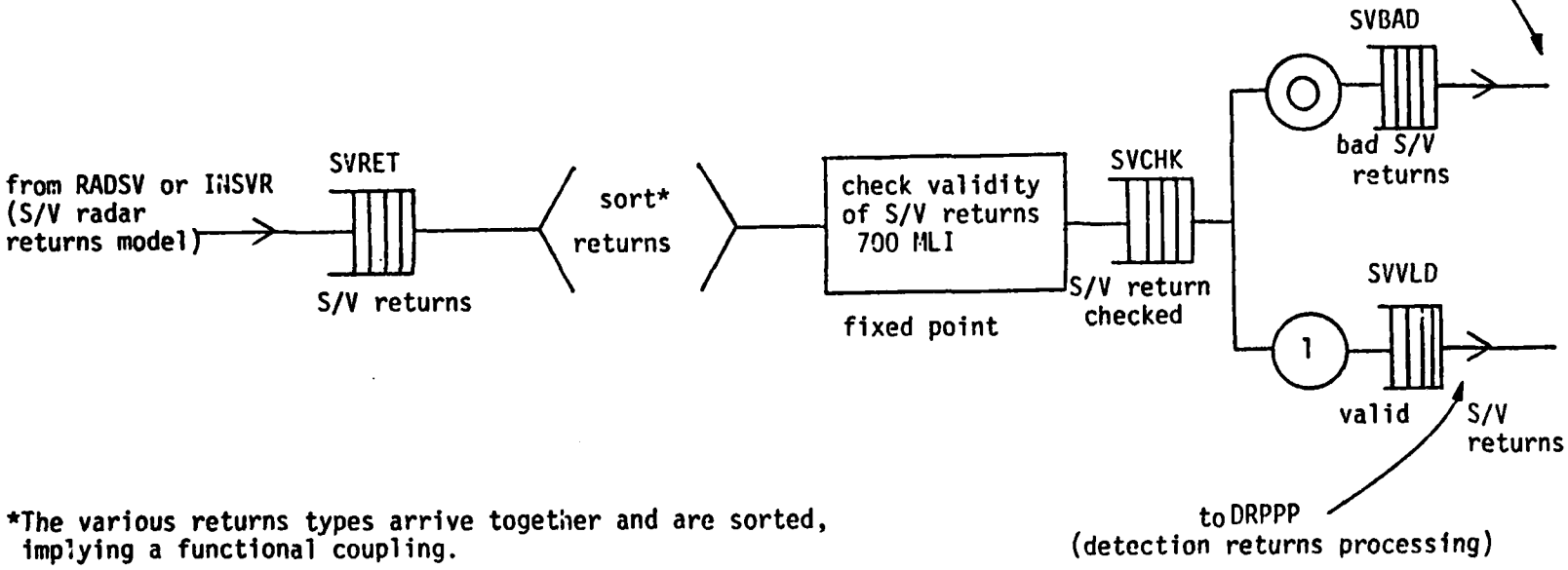
RADAR RETURNS ASSIMULATION FUNCTION

Primitive: VLDSV

Validity checking of S/V returns

Radar returns assimilation function

Bad returns may be identified and sent back for rescheduling. At present we take all returns to be valid. As we do not model processing of SVBAD, this is taken as a system output.



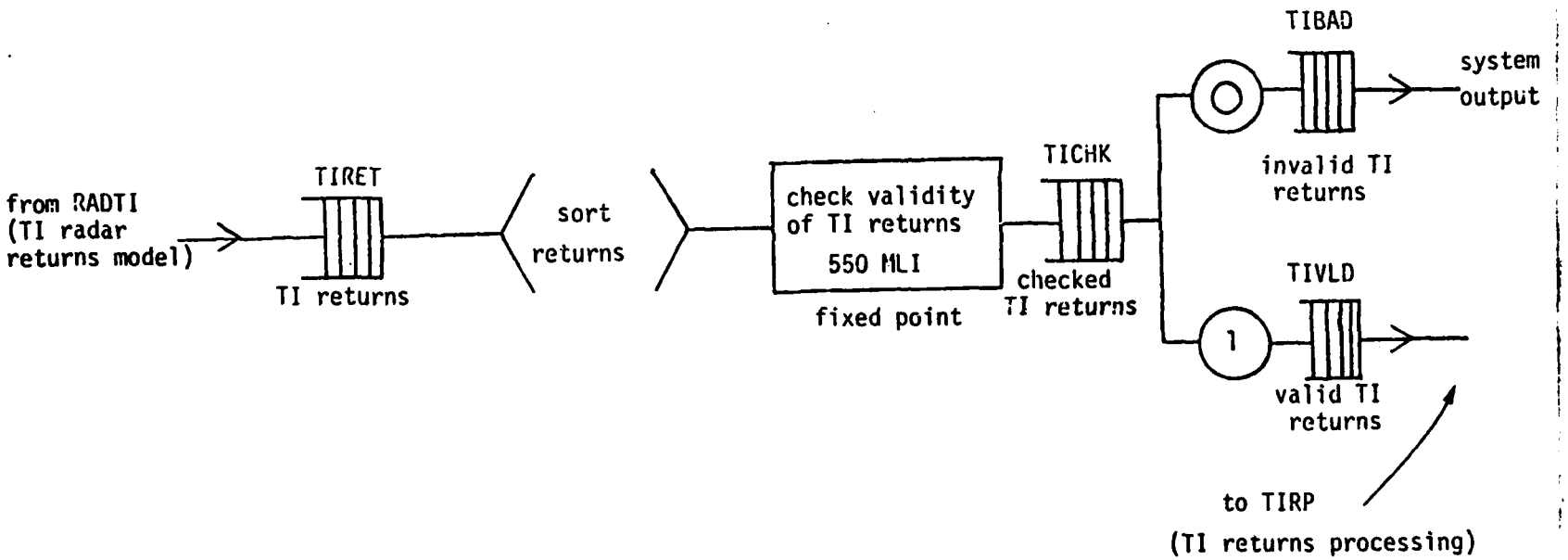
*The various returns types arrive together and are sorted, implying a functional coupling.

Use zero delay default for "sort returns"

Primitive: VLDTI

Validity checking of TI returns

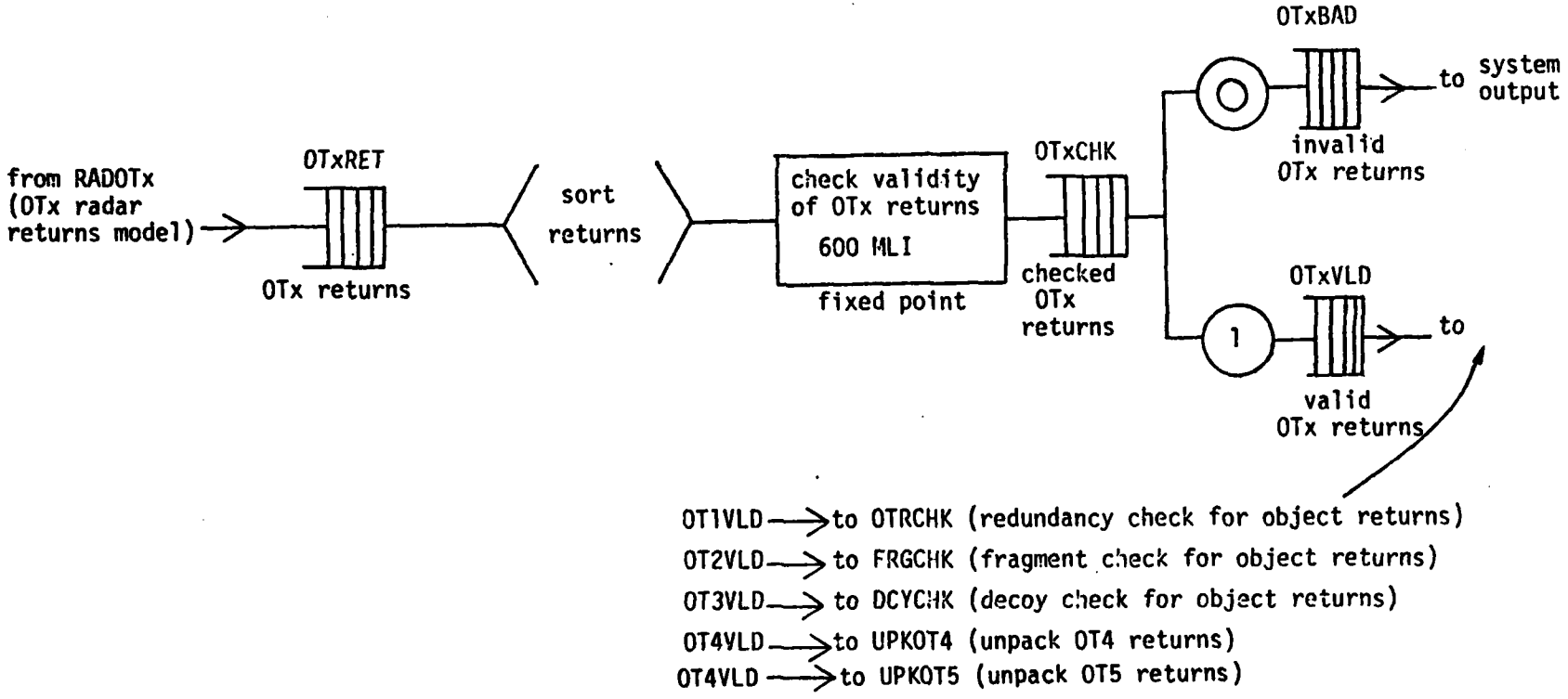
Radar returns assimilation function



Primitive: VLDOTx (x = 1,2,3,4 ,5)

Validity checking of OTx returns

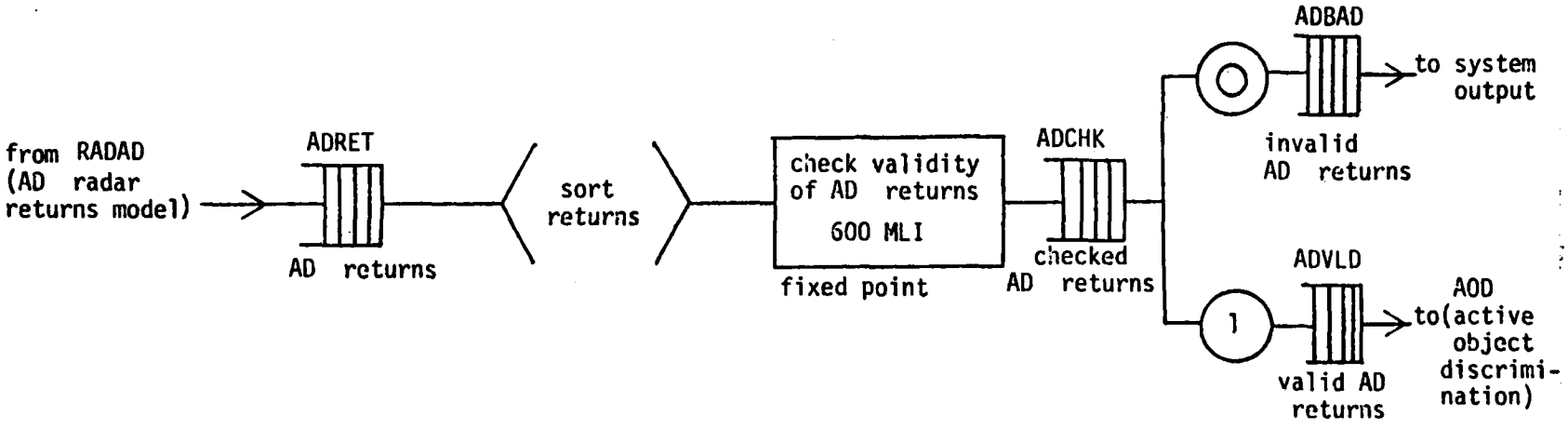
Radar returns assimilation funcion



Primitive: VLDAD (x = 1,2)

Validity checking of AD returns

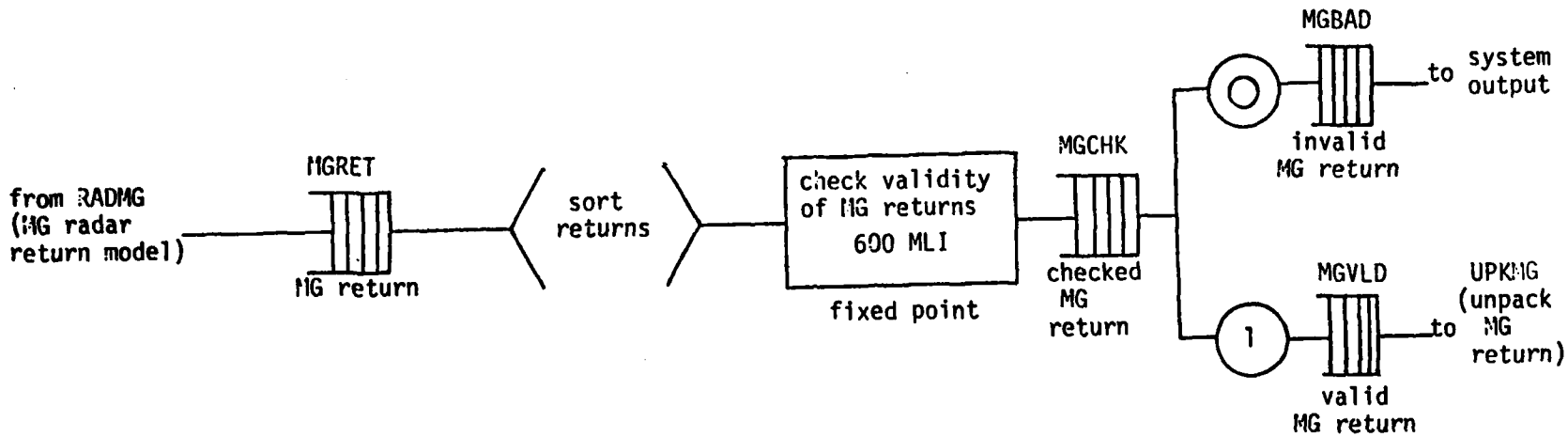
Radar returns assimilation function

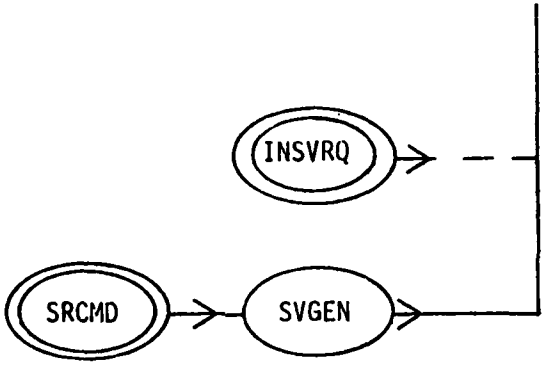


Primitive: VLDMG

Validity checking of MG returns

Radar returns assimilation function





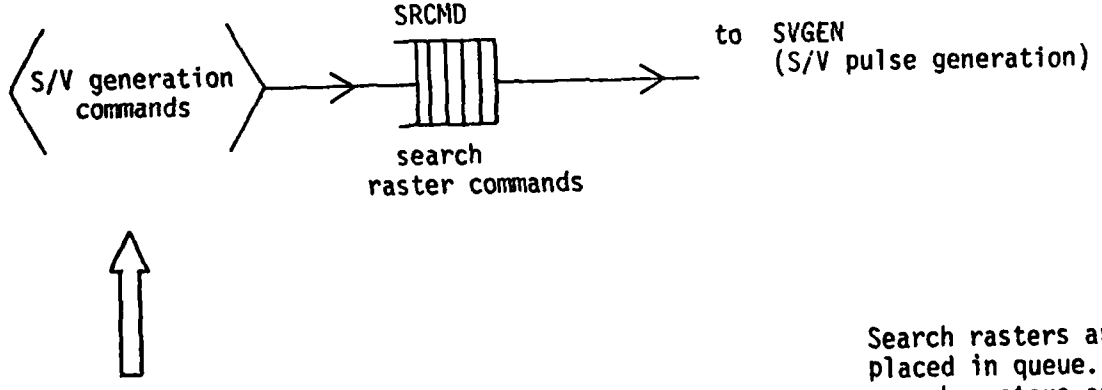
SEARCH RASTER GENERATION FUNCTION

Primitive: SRCMD

Search raster commands

Input

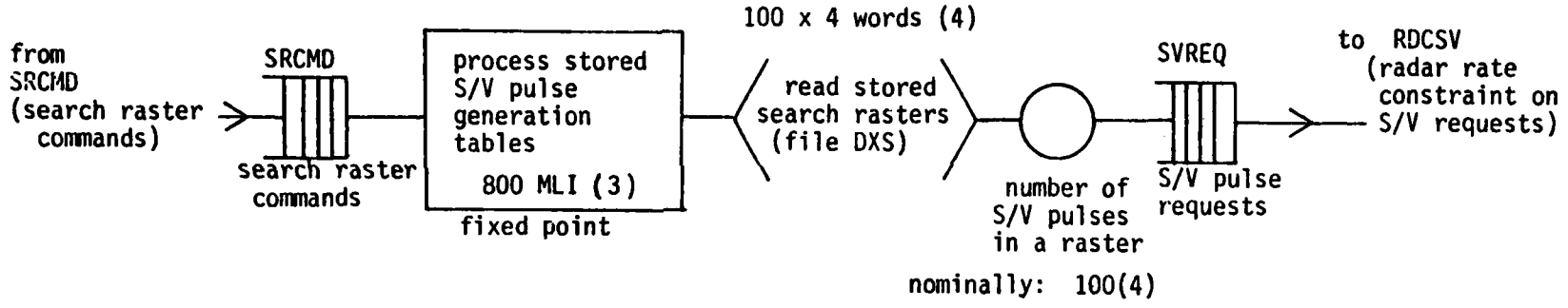
zero delay



Search rasters are periodically placed in queue. In addition, new search rasters are occasionally added and old ones deleted. This primitive models the total number of search raster commands, both those periodically generated and new ones.

Primitive: SVGEN

S/V pulse request generation



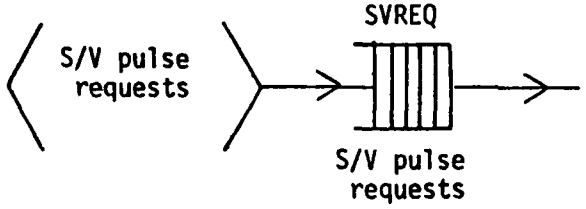
Search rasters are precomputed and stored in tables (file DXS). In the existing implementation, not all active rasters can be locally stored, so the active rasters are cycled by destructive reads. In addition, commands for new rasters are also processed.

Primitive: INSVRQ

S/V pulse request generation

input

zero-delay



to RDCSV
(radar rate constraint
on S/V requests)

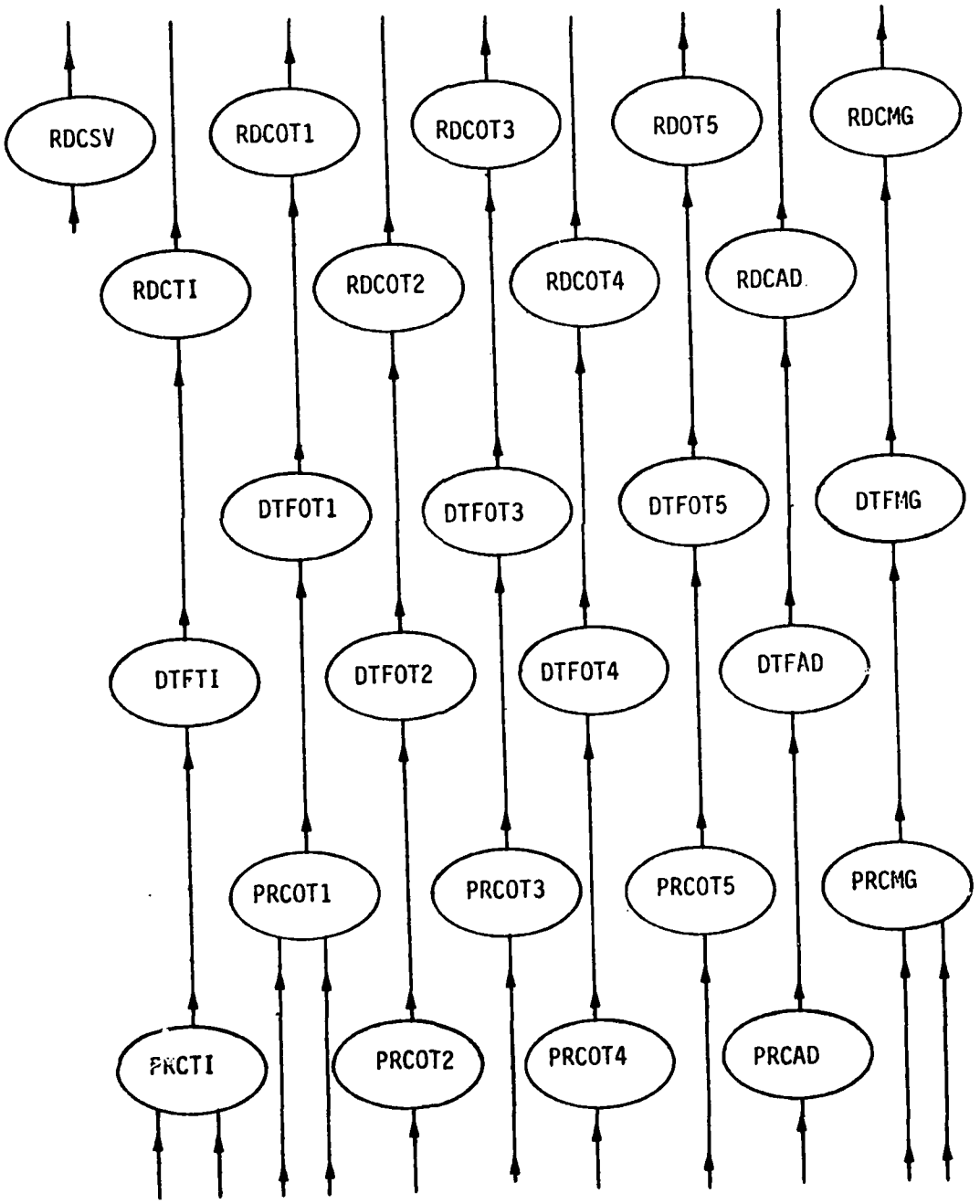
This input primitive provides direct control over the rate of S/V pulse requests.

When used, the output of primitive SVGEN is a system output (does not drive subsequent processing).

scenario input
analytic model
uniform rate in
the range 55-2000 pps (5)

MACROSCHEDULING

FUNCTION

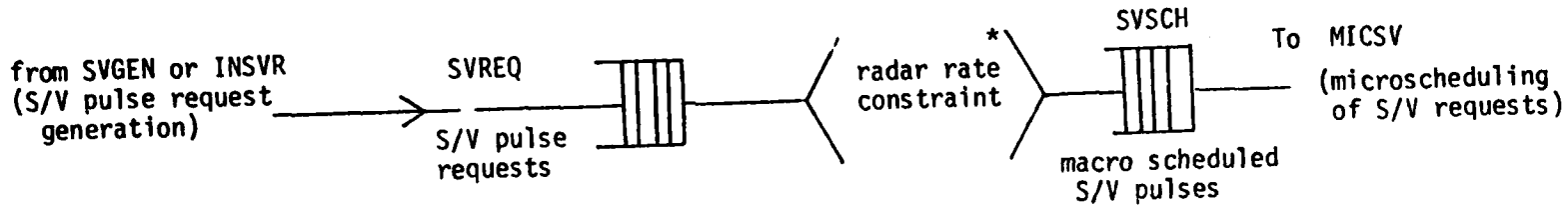


Primitive: RDCSV

Radar rate constraint on S/V requests

Radar scheduling function

May be zero delay (when constraint not active)

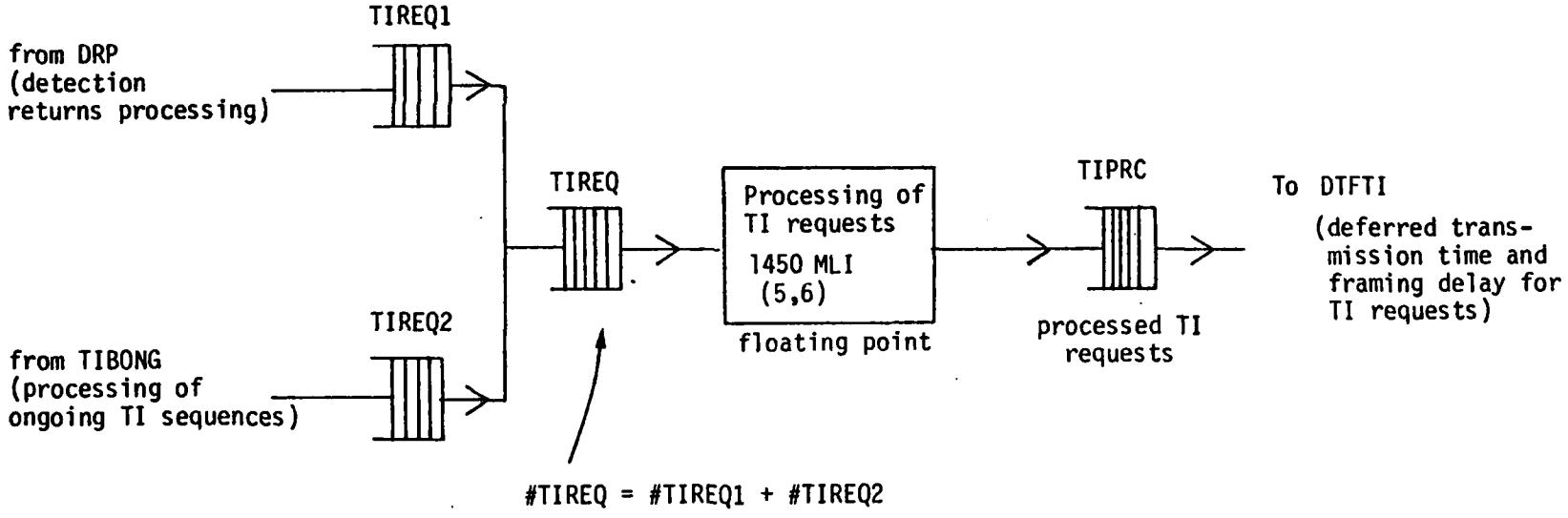


*The various pulse requests are coupled through the radar rate constraint model.

Primitive: PRCTI

Processing of TI requests

Radar scheduling function

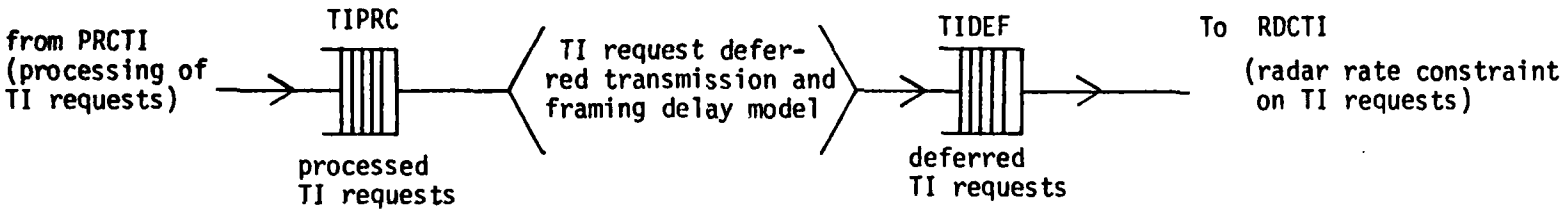


Using nominal transmission times and TI state information, TI requests are processed to produce radar parameters.

Primitive: DTFTI

Deferred transmission time and framing delay for TI requests

Radar scheduling function



Delay is modeled by a pipeline:

$$\text{rate} = \frac{\#\text{ in TIPRC queue}}{\text{total delay}}$$

$$\text{total delay} = \delta_{TI} + f$$

δ_{TI}: deferral time to achieve nominal transmission time

f: framing delay, where f is the frame size

take $\delta_{TI} = 0$

$f = 10\text{ms}$

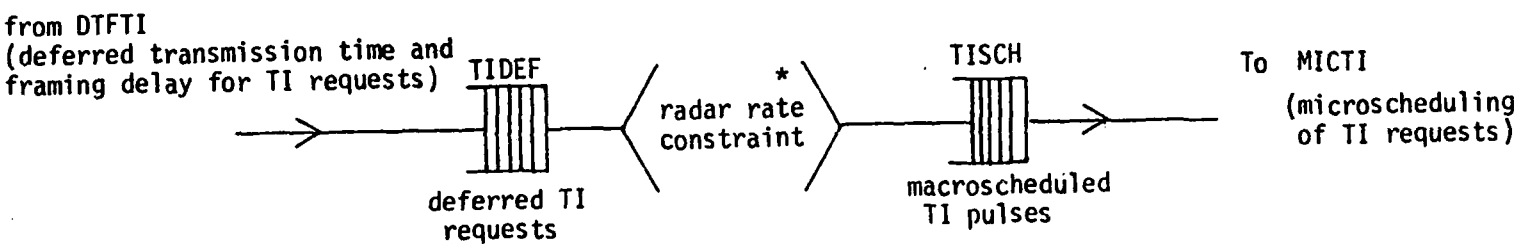
δ_{TI} will be determined as the additional delay imposed to achieve nominal PTP response time. In heavy attack, δ_{TI} may be zero, yet the actual PTP response time may exceed the goal.

Primitive: RDCTI

Radar rate constraint on TI requests

Radar scheduling function

May be zero delay (when constraint not active)

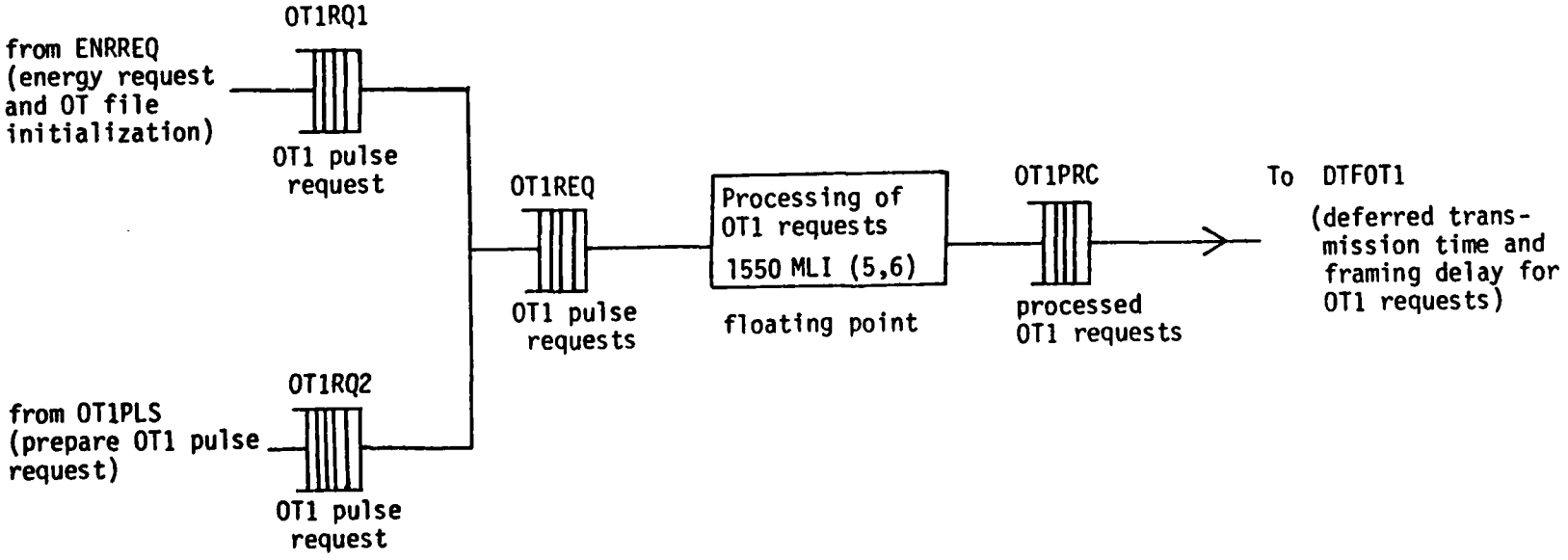


*The various pulse requests are coupled through the radar rate constraint model.

Primitive: PRCOT1

Process OT1 pulse request

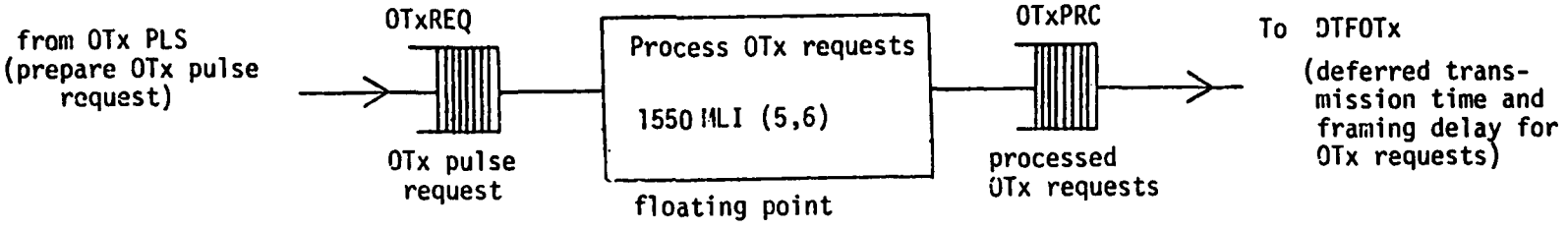
Radar scheduling function



Primitive: PRCOTx (x = 2,3,4,5)

Process OTx pulse request

Radar scheduling function

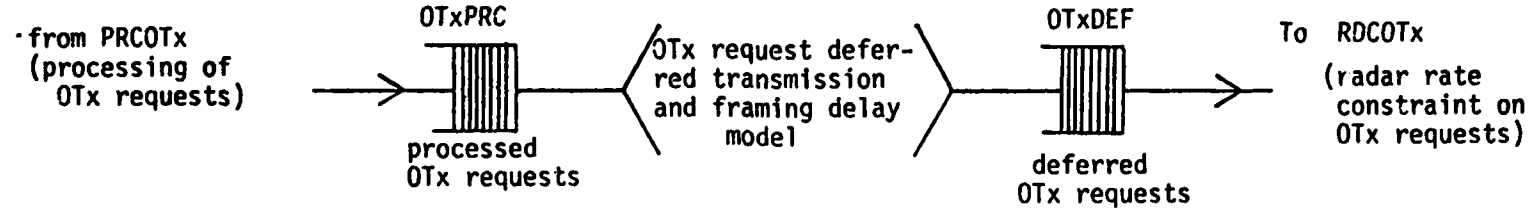


Note: PRCOT4 may be zero delay.

Primitive: DTFOTx (x = 1,2,3,4,5)

Deferred transmission time and framing delay for OTx requests

Radar scheduling function



Delay is modeled by a pipeline:

$$\text{rate} = \frac{\#\text{ in OTxPRC queue}}{\text{total delay}}$$

take $\delta_{OTx} = 0$

$f = 10\text{ms}$

$$\text{total delay} = \delta_{OTx} + f$$

δ_{OTx}: deferred time to achieve nominal transmission time

f: framing delay, where f is the frame size

Primitive: RDCOTx (x = 1,2,3,4,5)

Radar rate constraint on OTx requests

Radar scheduling function

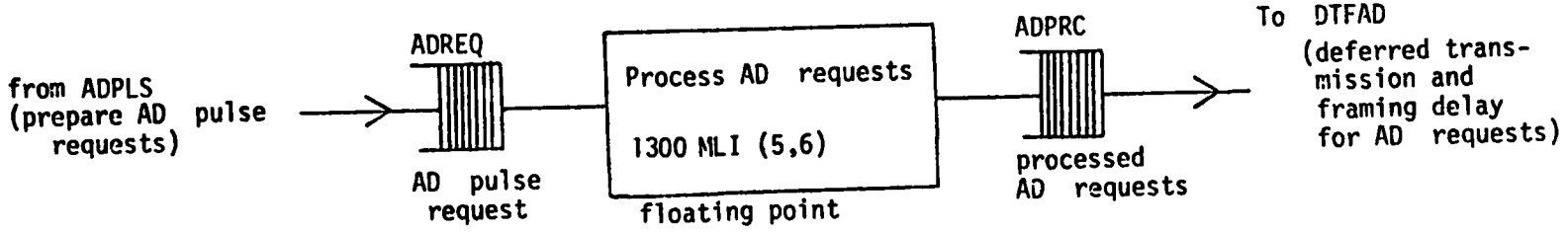
May be zero delay (when constraint not active)

*The various pulse requests are coupled through the radar rate constraint model.

Primitive: PRCAD

Process AD pulse request

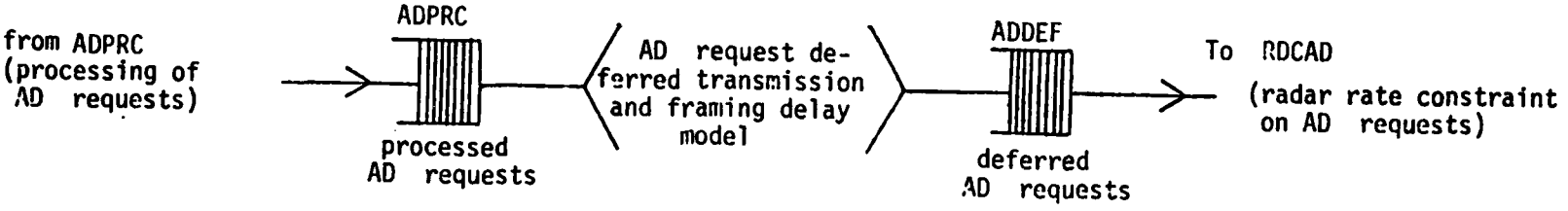
Radar scheduling function



Primitive: DTFAD

Deferred transmission time and framing delay for ADx requests

Radar scheduling function



Model

$$\text{rate} = \frac{\text{\# in ADxPRC queue}}{\text{total delay}}$$

$$\text{total delay} = \delta_{ADx} + f$$

$$\delta_{ADx} = 0$$

$$f = 10\text{ms}$$

Primitive: RDCAD

Radar rate constraint on AD requests

Radar scheduling function

May be zero delay (when constraint not active)

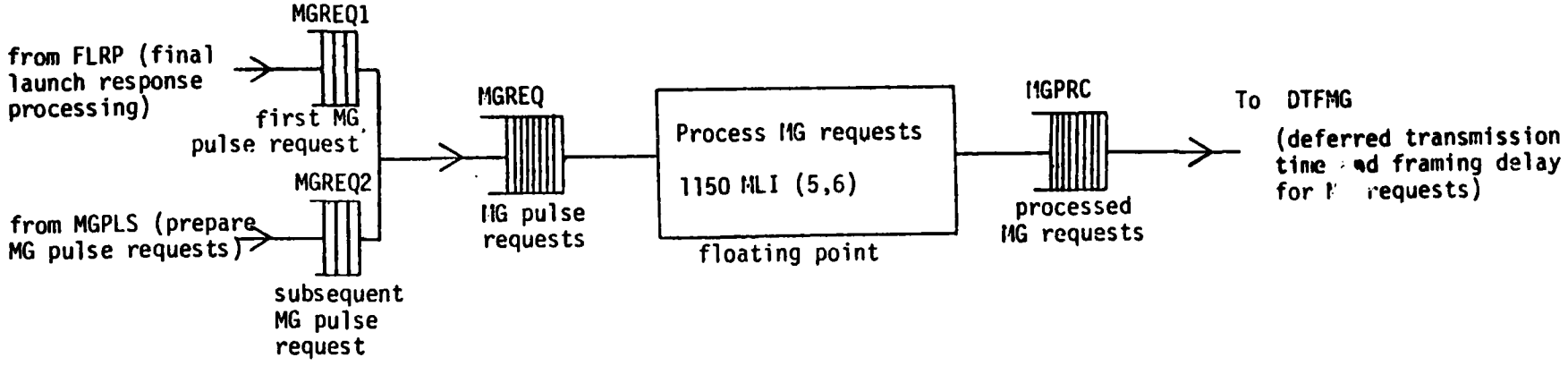
*The various pulse requests are coupled through the radar rate constraint model.

Primitive: PRCMG

Process MG pulse request

Radar scheduling function

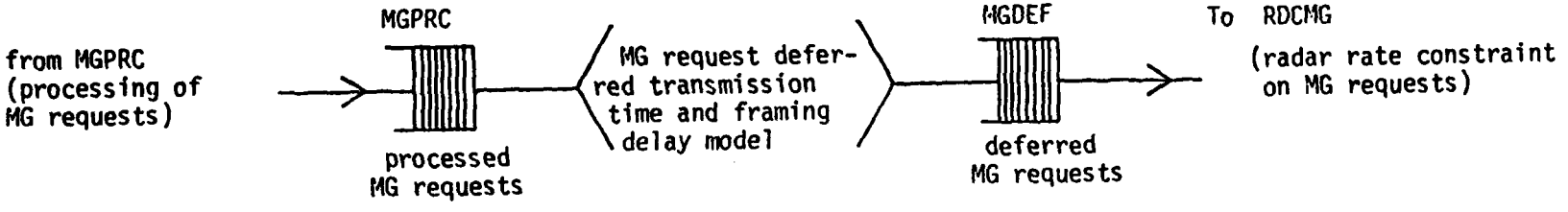
May be zero delay



Primitive: DTFMG

Deferred transmission time and framing delay for MG requests

Radar scheduling function



Model

$$\text{rate} = \frac{\text{\# in MGPRC queue}}{\text{total delay}}$$

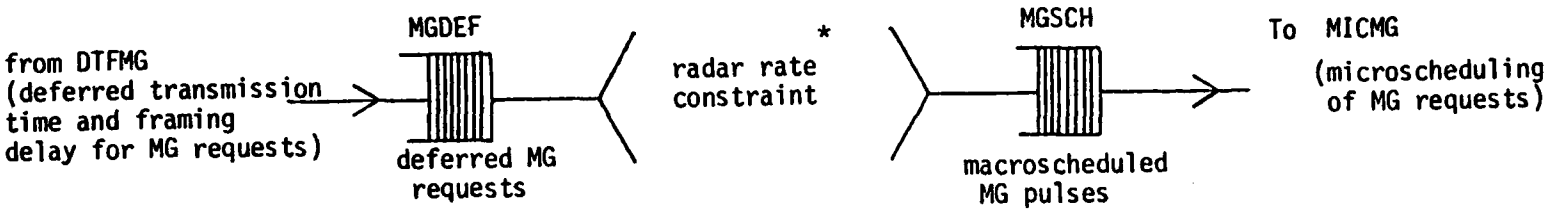
$$\text{total delay} = \delta_{MG} + f$$

Primitive: RDCMG

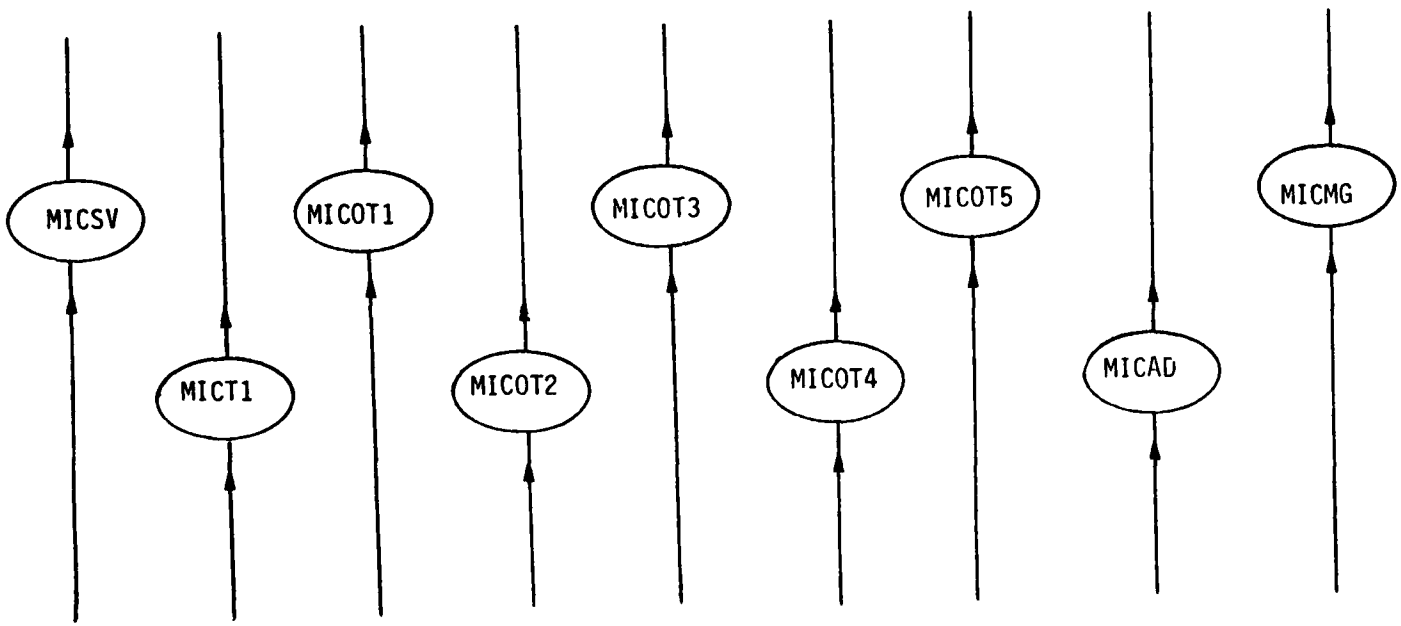
Radar rate constraint on MG requests

Radar scheduling function

May be zero delay (when constraint not active)



*The various pulse requests are coupled through the radar rate constraint model.

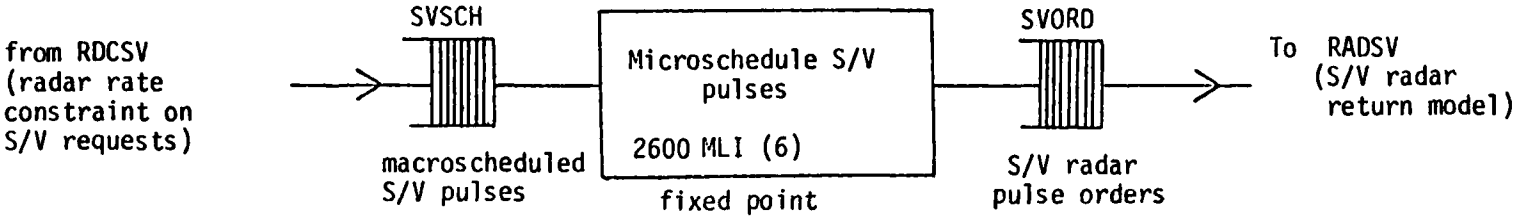


MICROSCHEDULING FUNCTION

Primitive: MICSV

Microscheduling of S/V requests

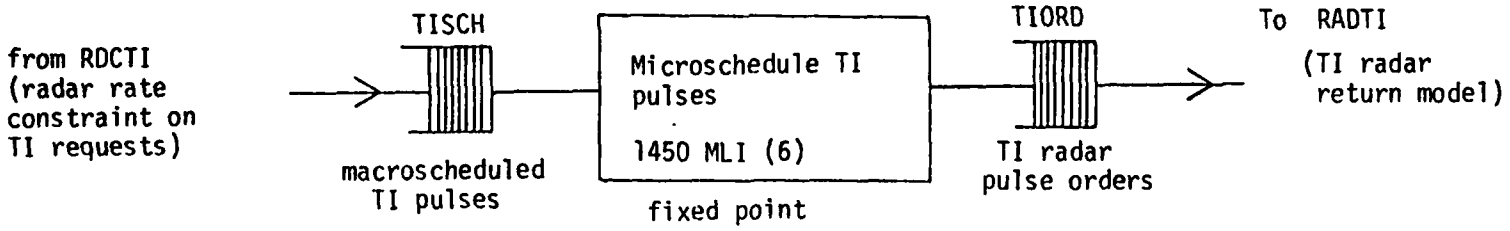
Radar scheduling function



Primitive: MICTI

Microscheduling of TI requests

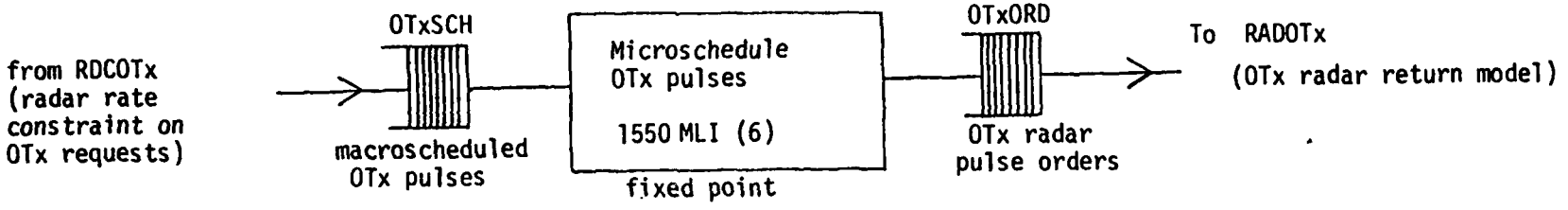
Radar scheduling function



Primitive: MICOTx (x = 1,2,3,4,5)

Microscheduling of OTx requests

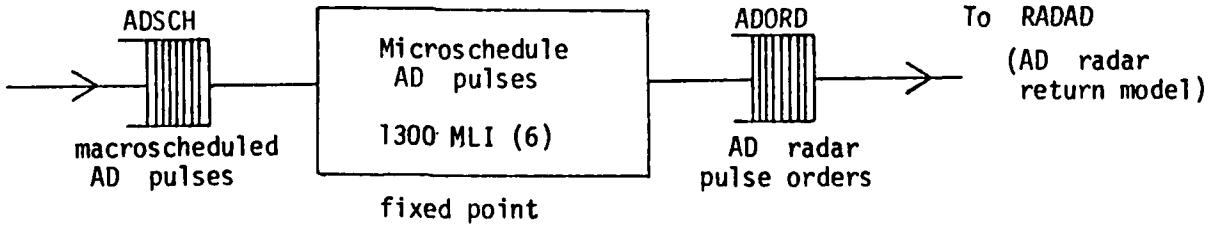
Radar scheduling function



Primitive: MICAD

Microscheduling of AD requests

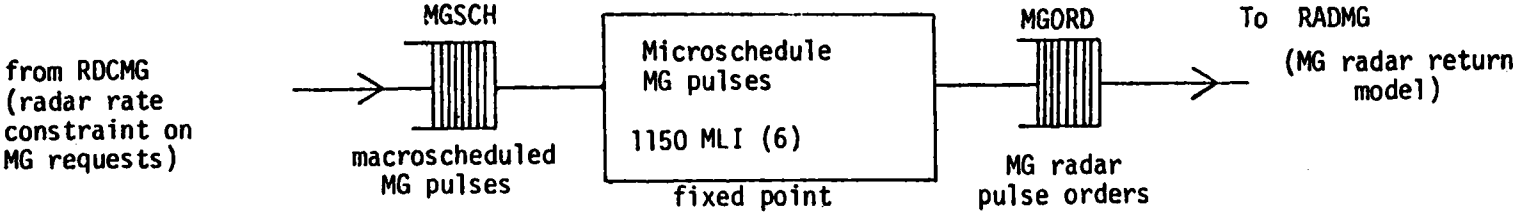
Radar scheduling function

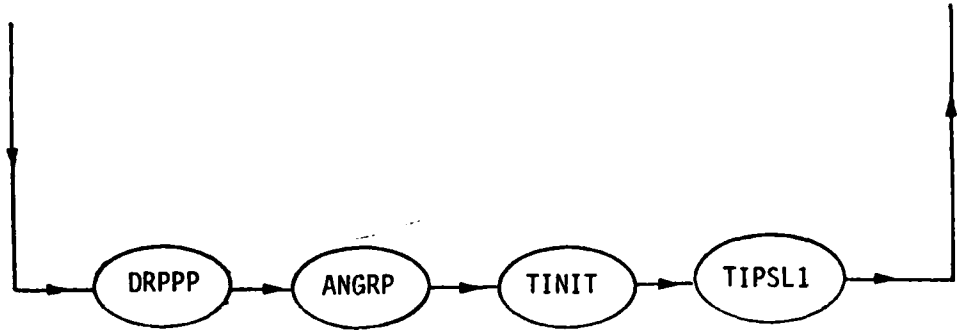


Primitive: MICMG

Microscheduling of MG requests

Radar scheduling function



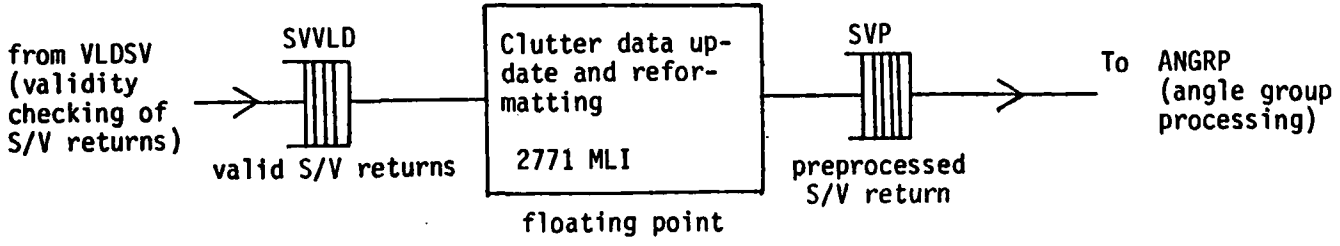


DETECTION RETURNS PROCESSING FUNCTION

Primitive: DRPPP

Detection returns preprocessing

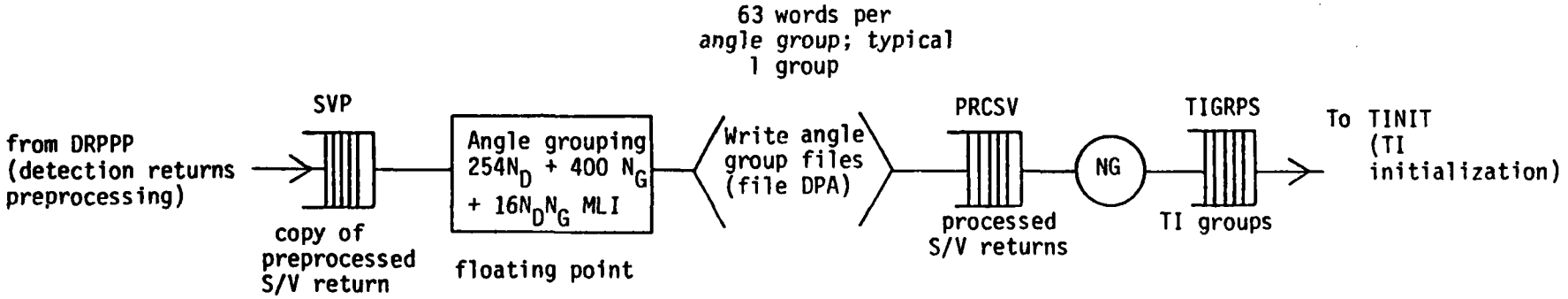
Detection returns processing function



Primitive: ANGRP

Angle group processing

Detection returns processing function



Nominal parameters

N_D: 3

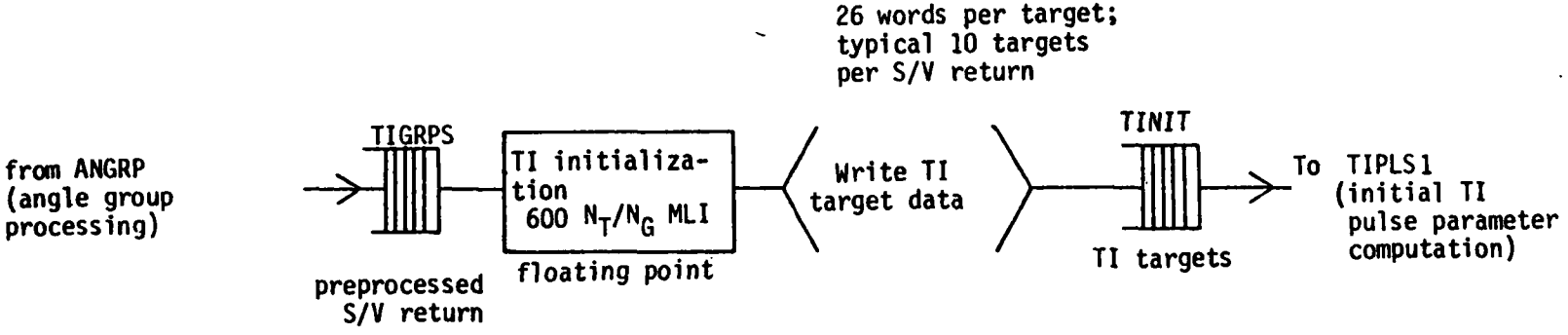
N_G: 1

Implies 1110 MLI

Primitive: TINIT

TI initialization

Detection returns processing function



Nominal parameters

N_G: 1

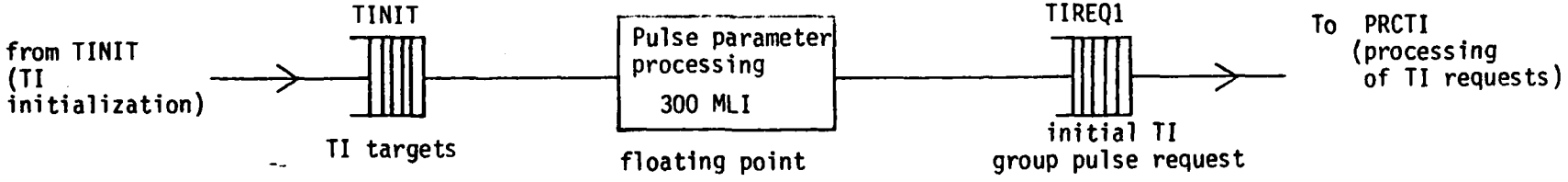
N_T: 3-1/3

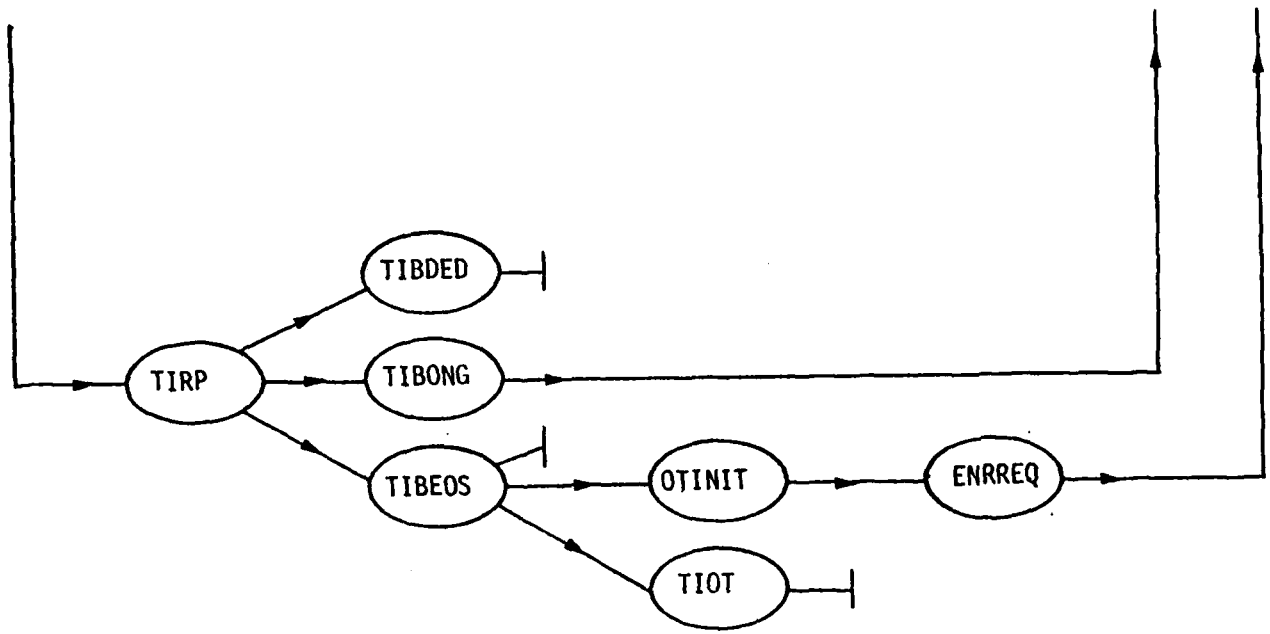
Implies 2000 MLI

Primitive: TIPLS1

Initial TI pulse parameter computation

Detection returns processing function



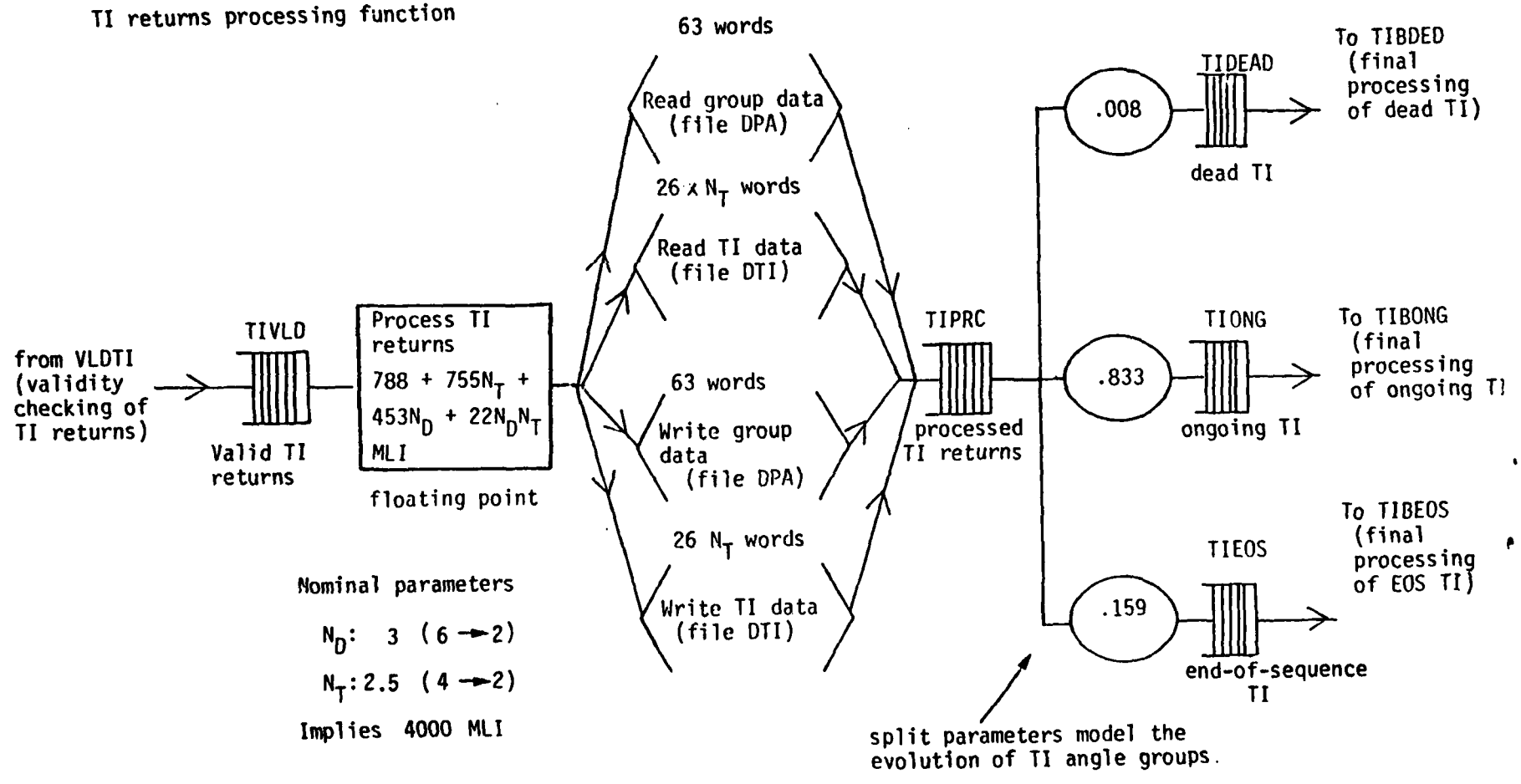


DETECTION RETURNS PROCESSING FUNCTION

Primitive: TIRP

TI returns processing

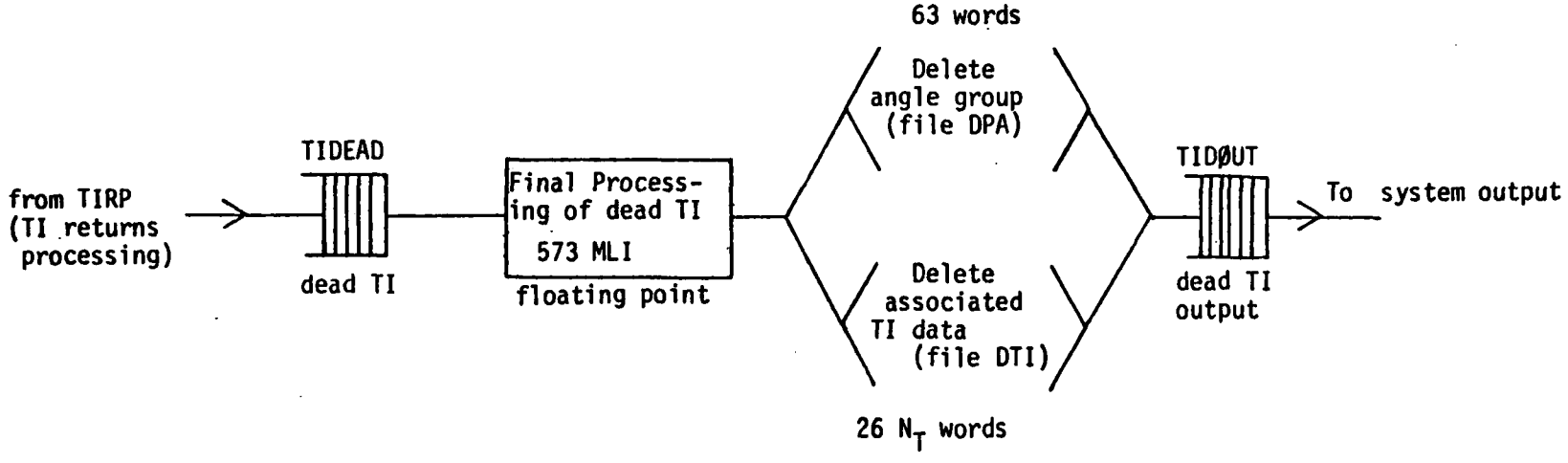
TI returns processing function



Primitive: TIBDED

Final processing of dead TI

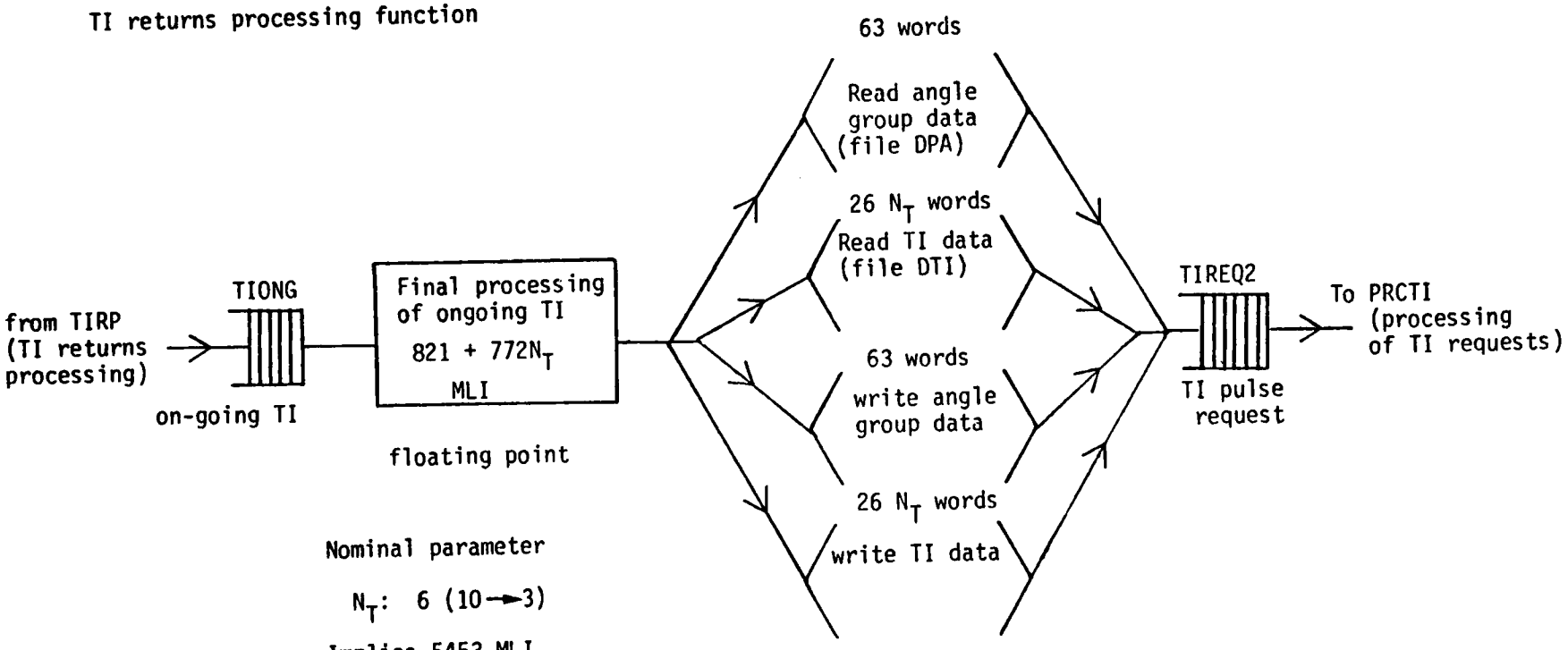
TI returns processing function



Primitive: TIBONG

Final processing of ongoing TI

TI returns processing function



Nominal parameter

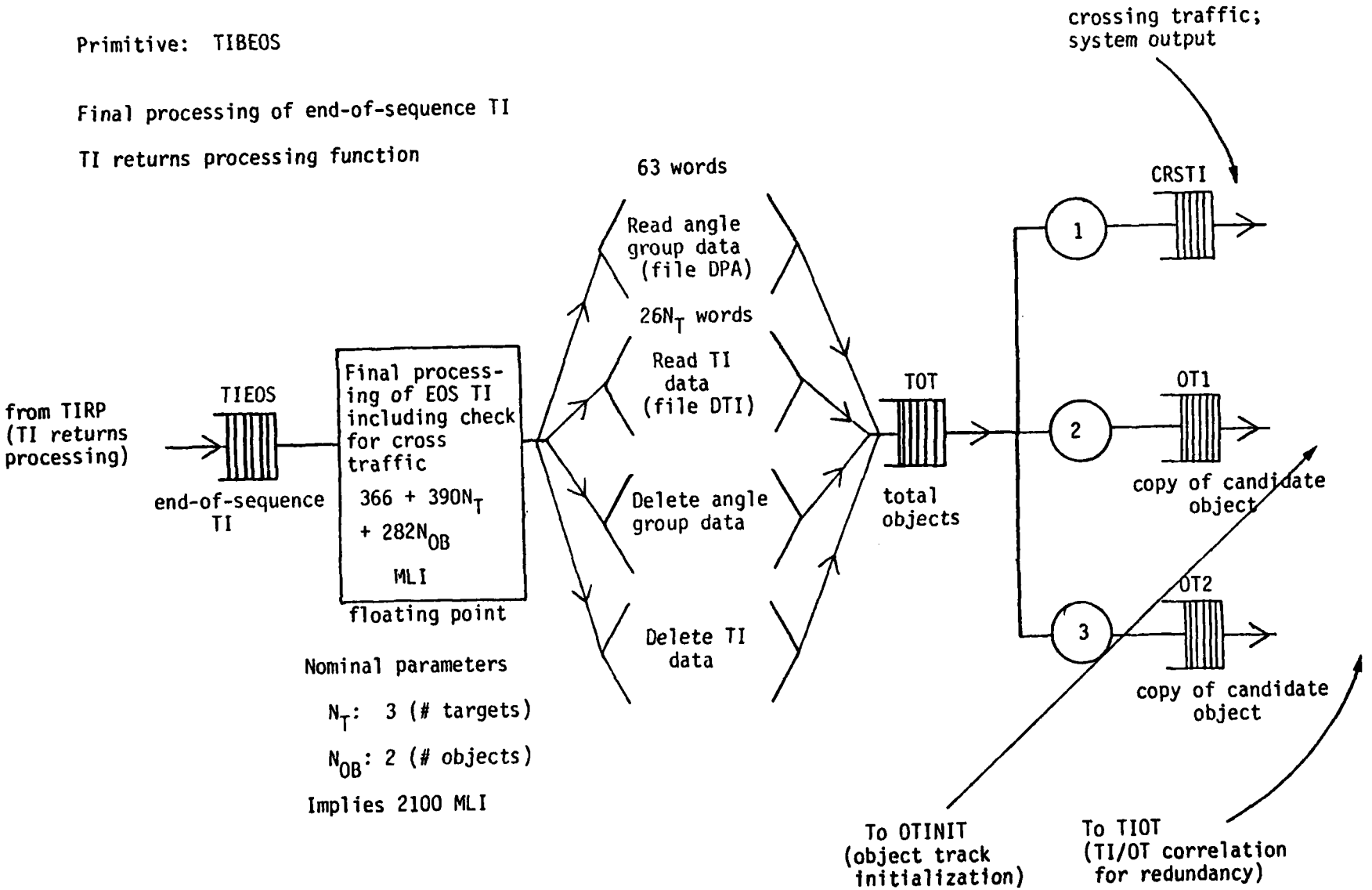
N_T: 6 (10→3)

Implies 5453 MLI

Primitive: TIBEOS

Final processing of end-of-sequence TI

TI returns processing function



Nominal parameters

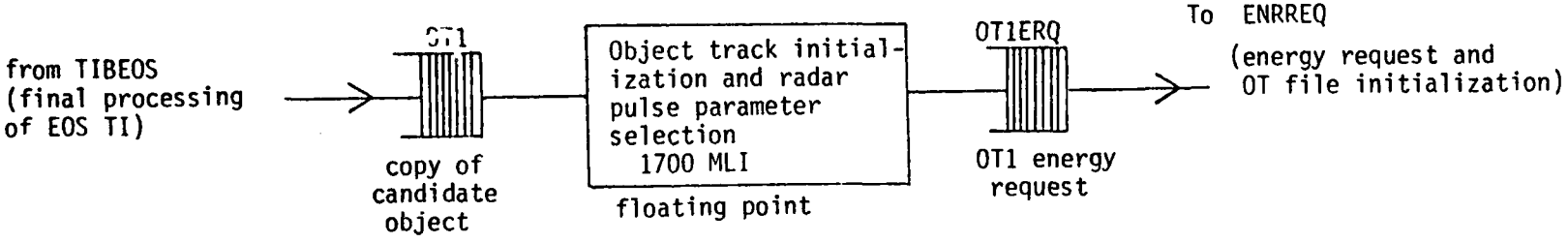
N_T: 3 (# targets)

N_{OB}: 2 (# objects)

Implies 2100 MLI

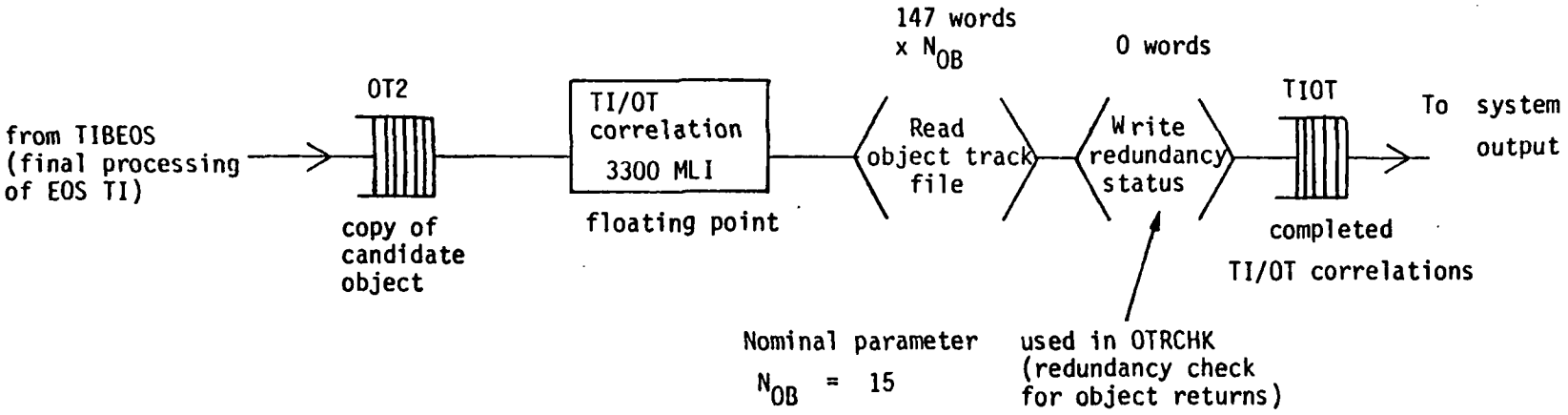
Primitive: OTINIT

Object track initialization



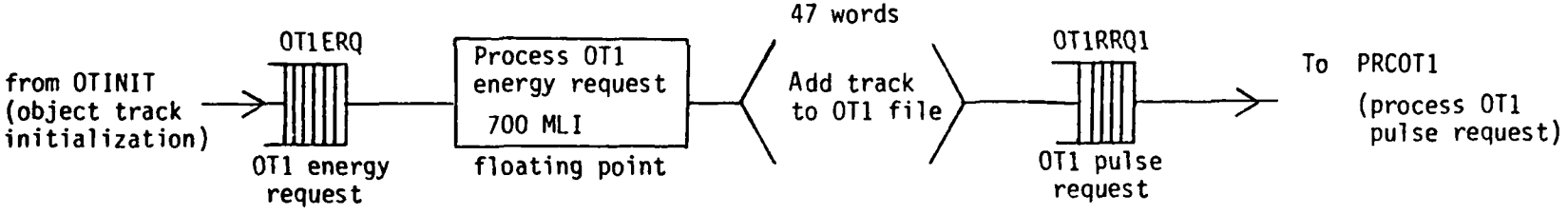
Primitive: TIOT

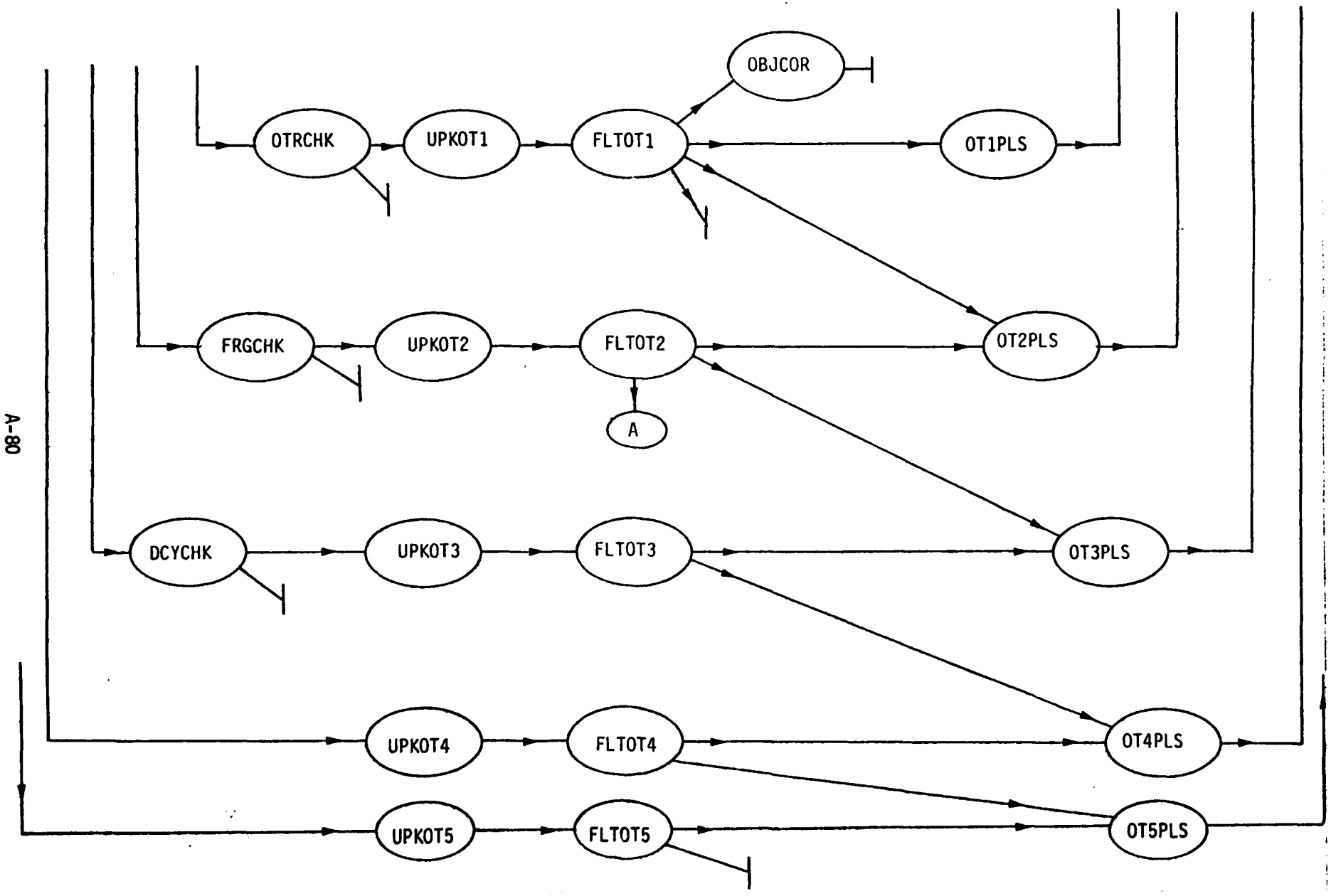
TI/OT Correlation for redundancy



Primitive: ENRREQ

Energy request and OT file initialization

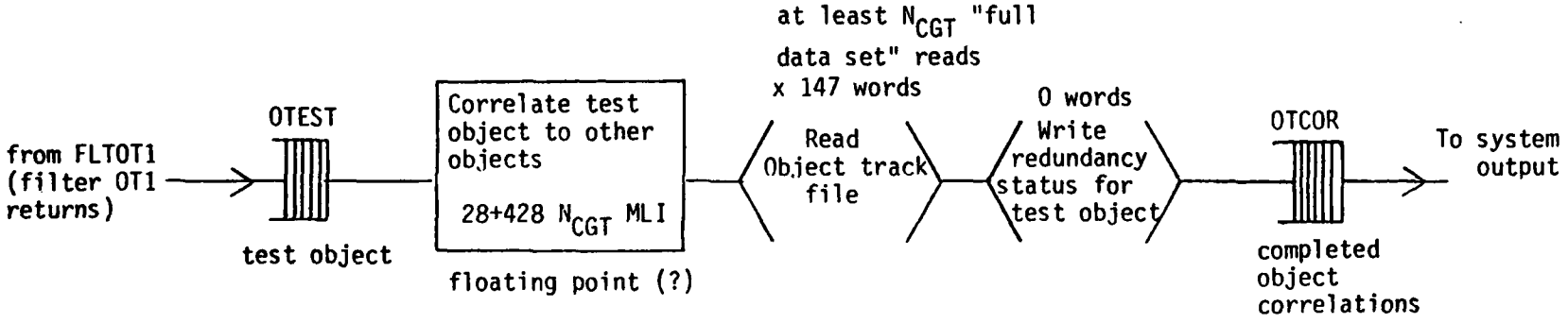



OBJCOR
OTRCHK
UPKOT1
FLTOT1
OT1PLS
FRGCHK
UPKOT2
FLTOT2
OT2PLS
A
DCYCHK
UPKOT3
FLTOT3
OT3PLS
UPKOT4
FLTOT4
OT4PLS
UPKOT5
FLTOT5
OT5PLS

Primitive: OBJCOR

Object correlation for redundancy

OT returns processing function



Nominal parameters

N_{CGT} (# full correlations)

= 10

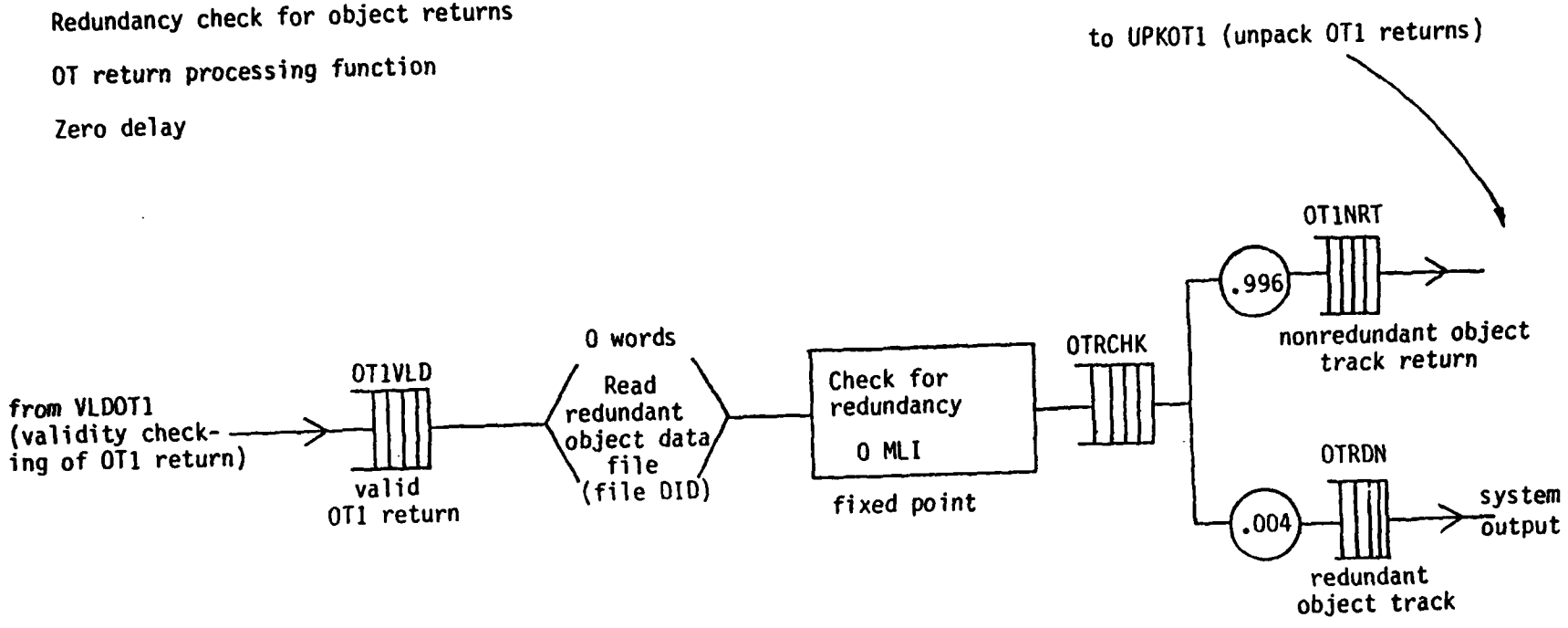
Implies 4308 MLI

Primitive: OTRCHK

Redundancy check for object returns

OT return processing function

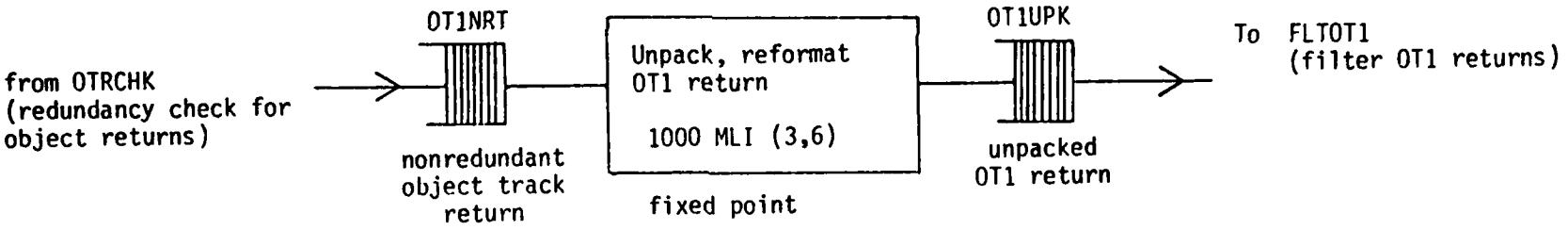
Zero delay



Primitive: UPKOT1

Unpack OT1 returns

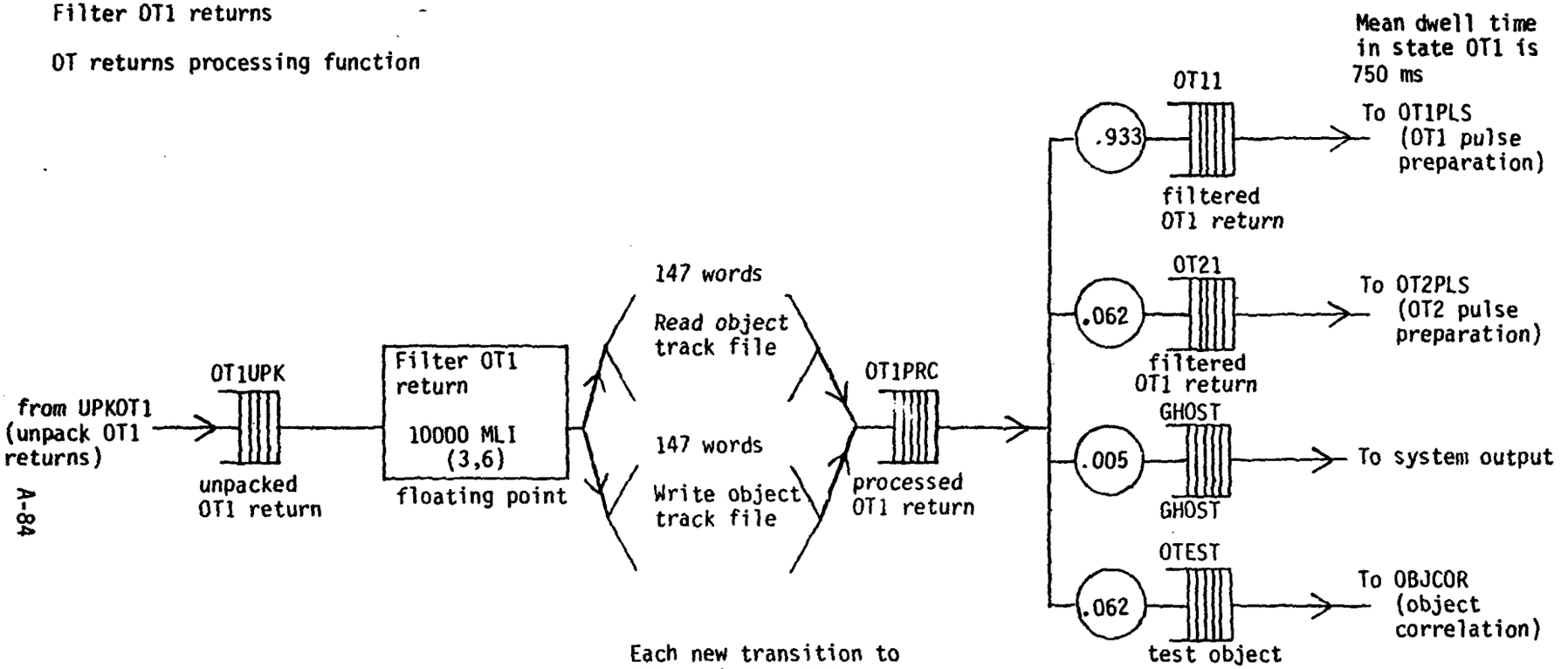
OT returns processing function



Primitive: FLTOT1

Filter OT1 returns

OT returns processing function



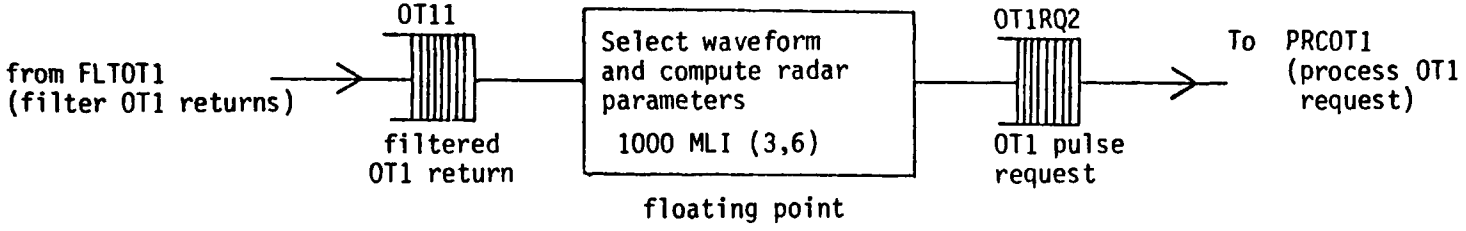
Each new transition to OT2 activates POD

Each *nonredundant*, nonghost track is subjected to *object correlation* once.

Primitive: OT1PLS

Prepare OT1 pulse

OT returns processing function

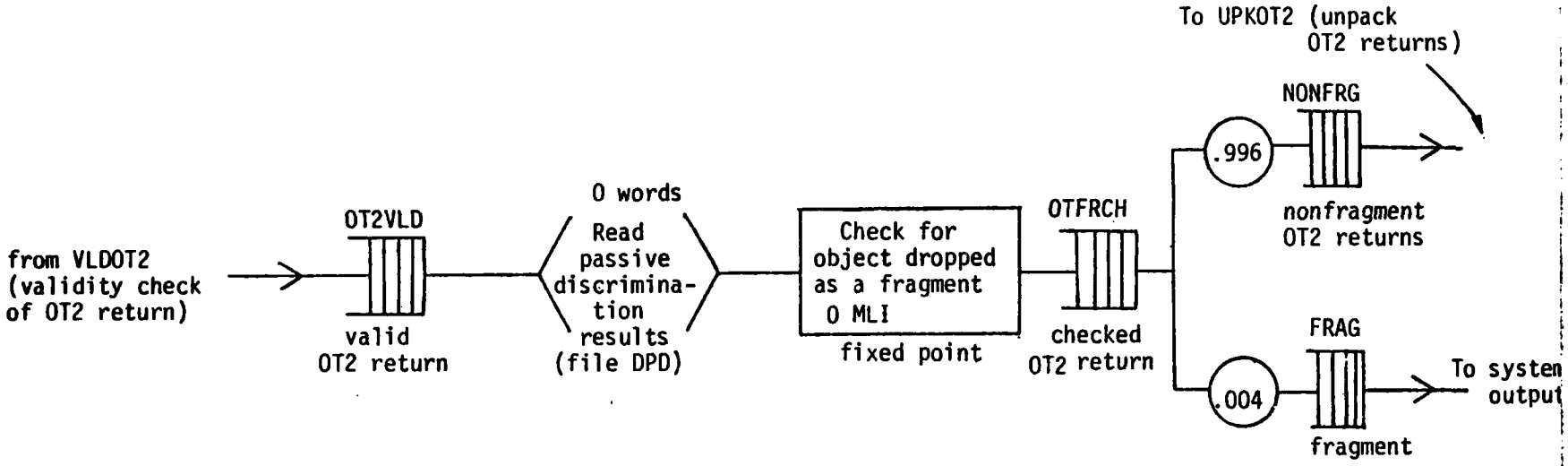


Primitive: FRGCHK

Fragment check for object returns

OT returns processing function

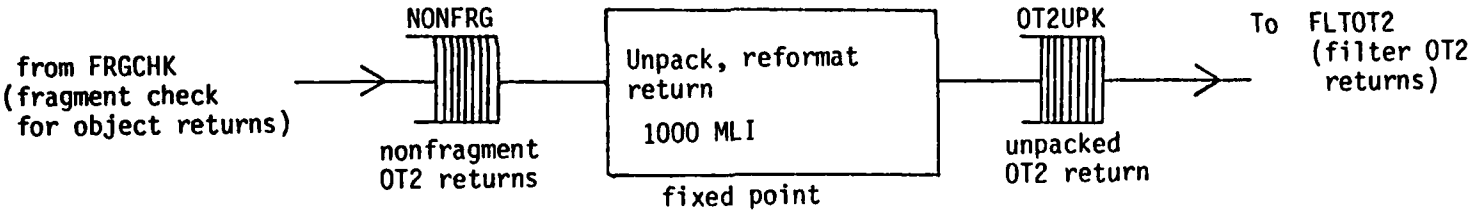
zero-delay



Primitive: UPKOT2

Unpack OT2 returns

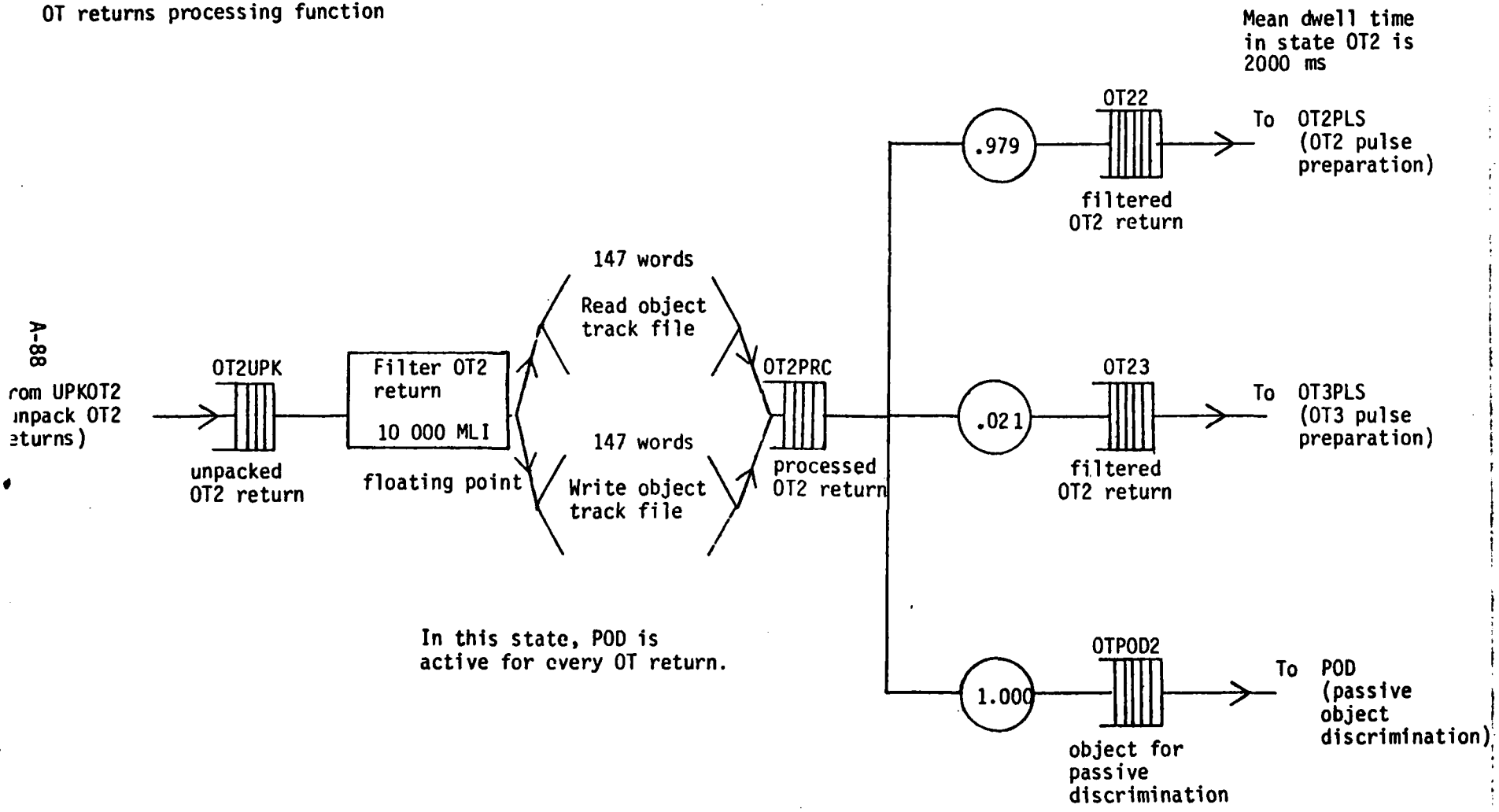
OT returns processing function



Primitive: FLTOT2

Filter OT2 returns

OT returns processing function

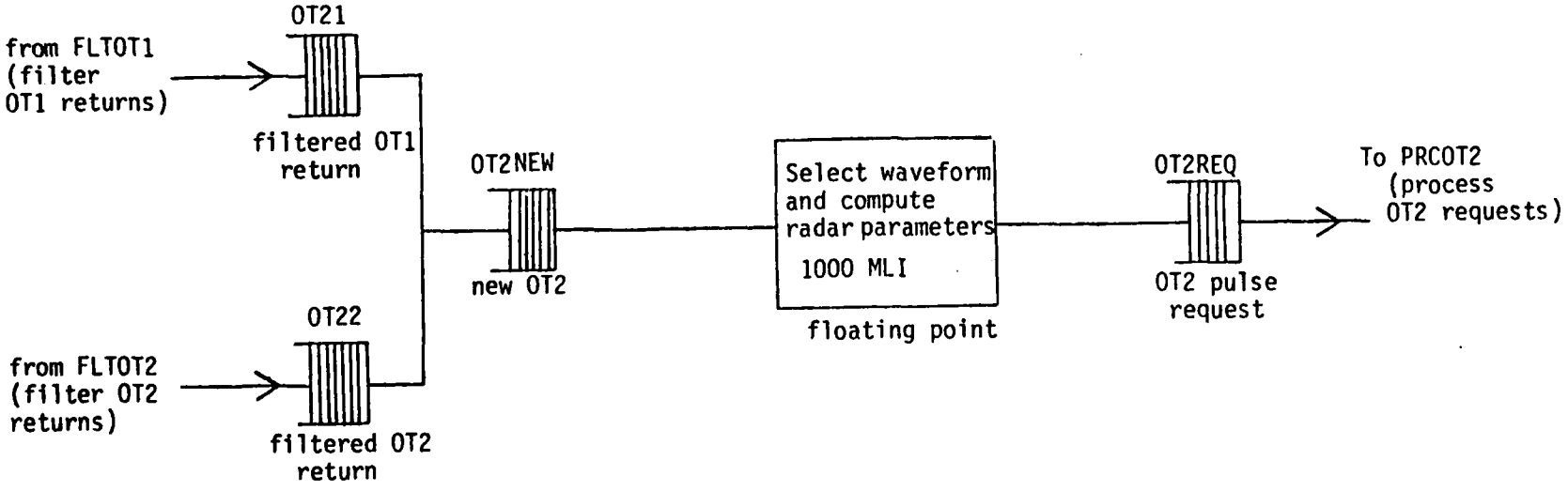


In this state, POD is active for every OT return.

Primitive: OT2PLS

Prepare OT2 pulse request

OT returns processing function

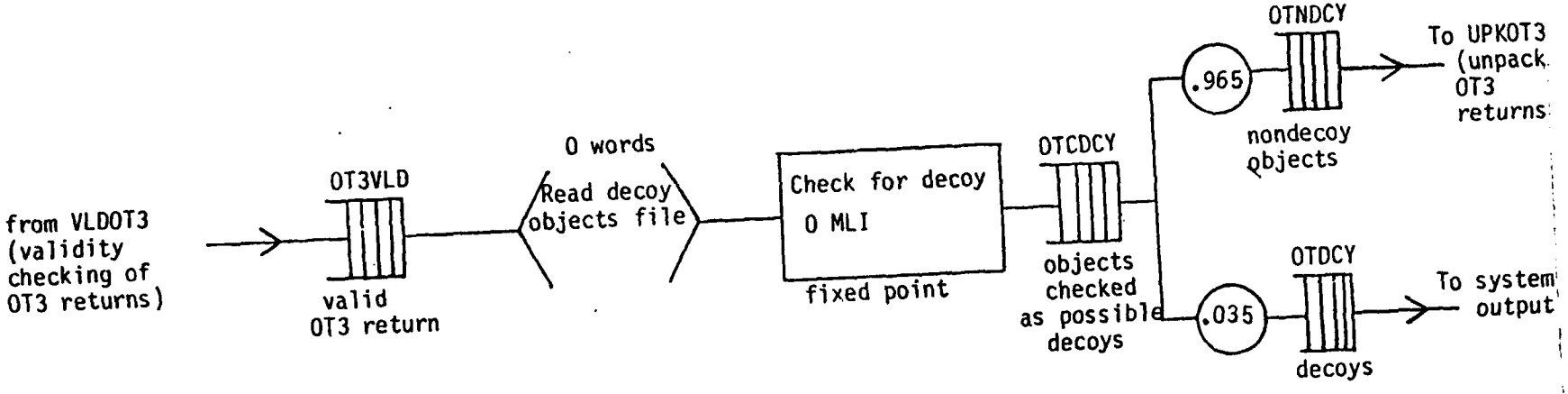


Primitive: DCYCHK

Decoy check for object returns

OT returns processing function

zero delay

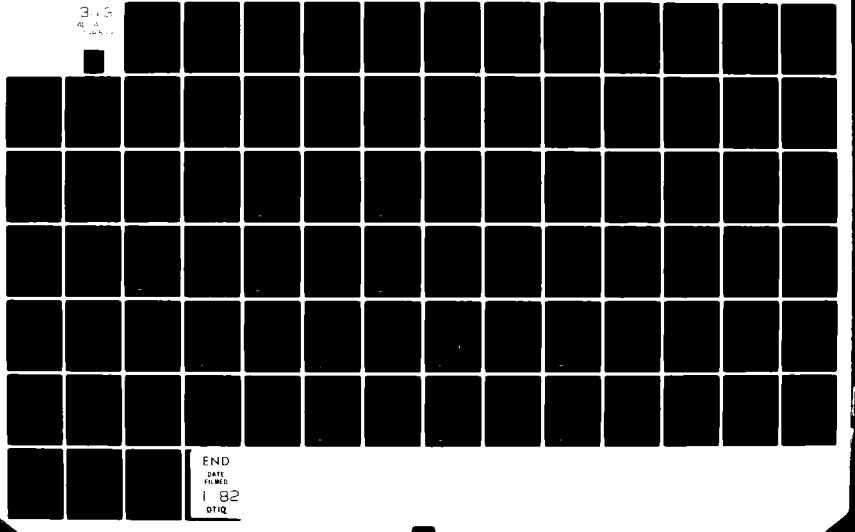


AD-A108 589 VERAC INC SAN DIEGO CA F/G 15/3.1

AN ANALYSIS OF MMCS NETWORK ARCHITECTURES TO SUPPORT THE DATA P--ETC(U)

DEC 80 J C TIERNAN DAS660-80-C-0017

UNCLASSIFIED R-008-80 NL

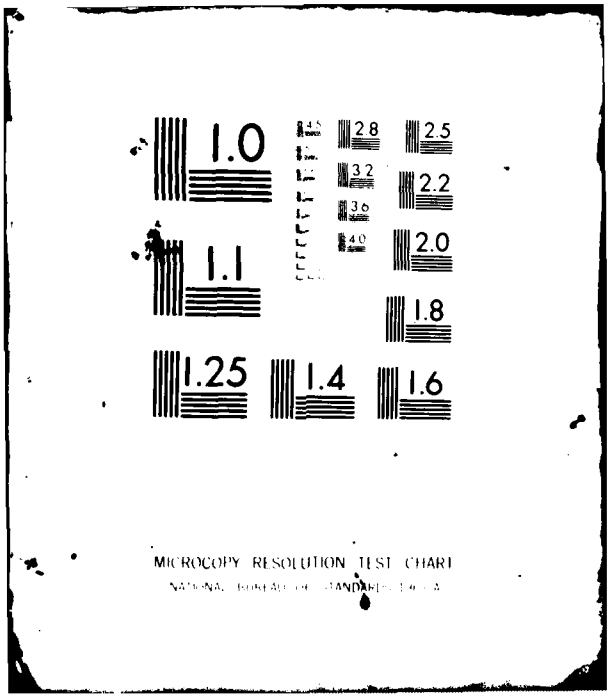


END

DATE FILMED

1 82

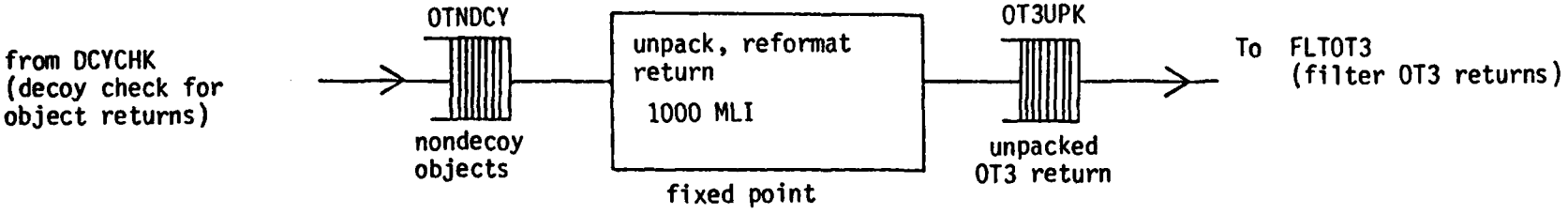
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART

Primitive: UPKOT3

Unpack OT3 returns

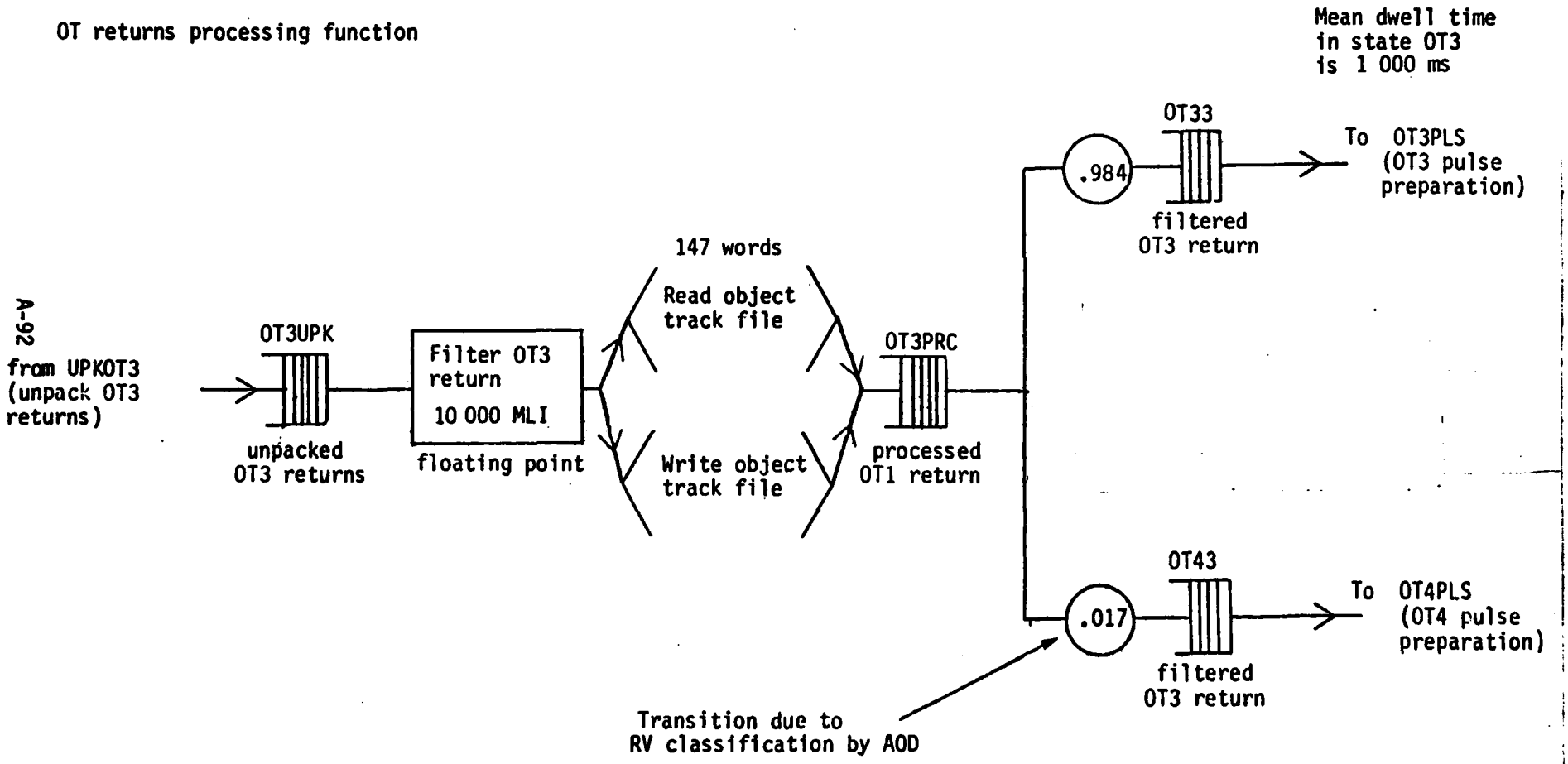
OT returns processing function



Primitive: FLTOT3

Filter OT3 returns

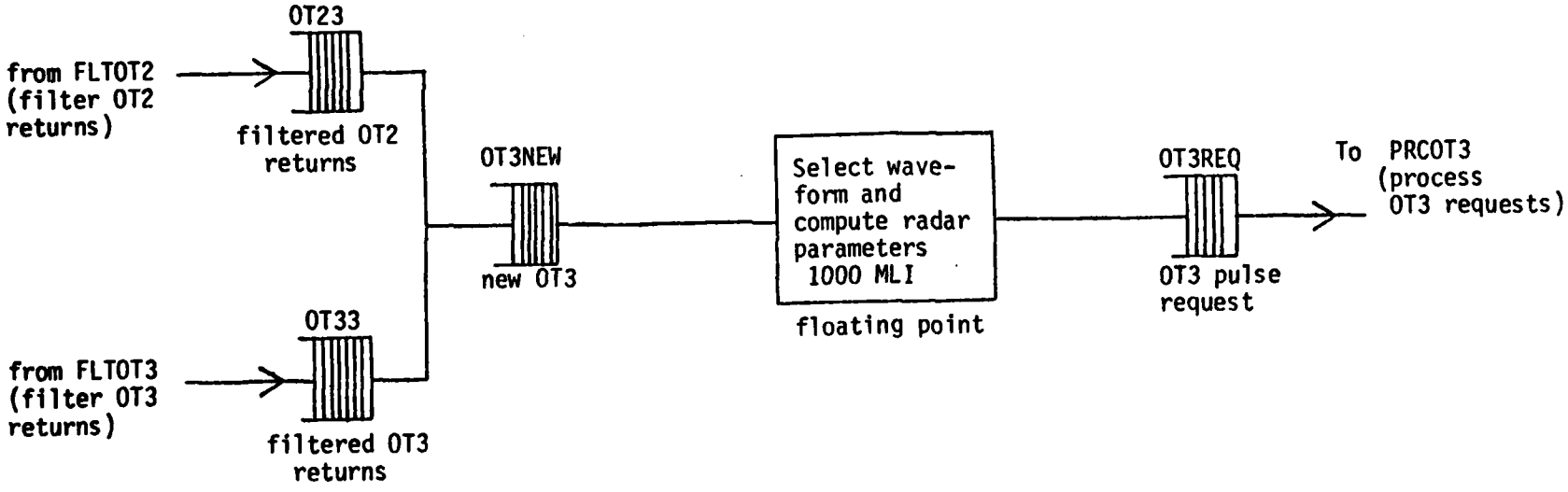
OT returns processing function



Primitive: OT3PLS

Prepare OT3 pulse request

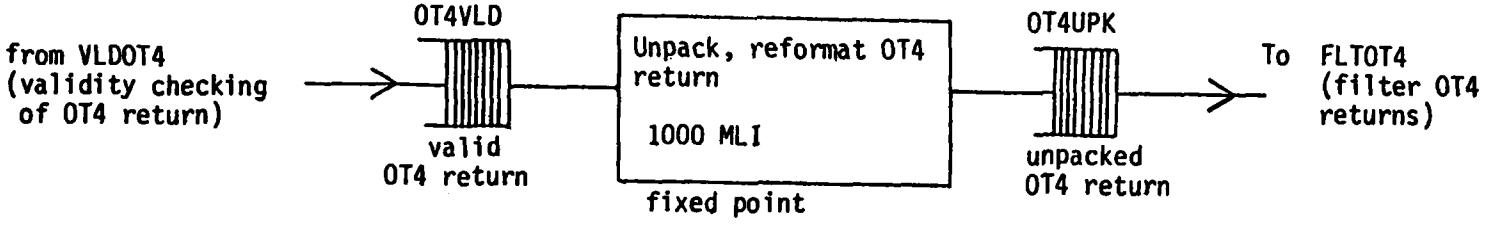
OT returns processing function



Primitive: UPKOT4

Unpack OT4 returns

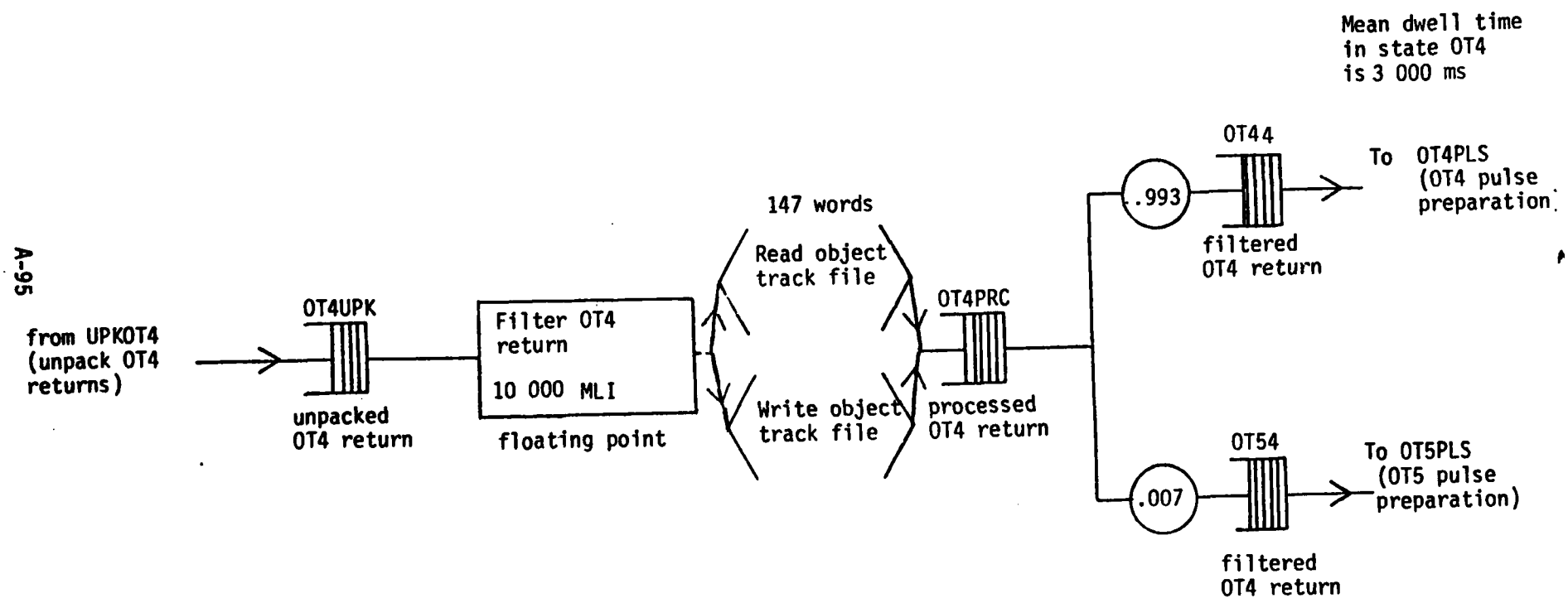
OT returns processing function



Primitive: FLTOT4

Filter OT4 returns

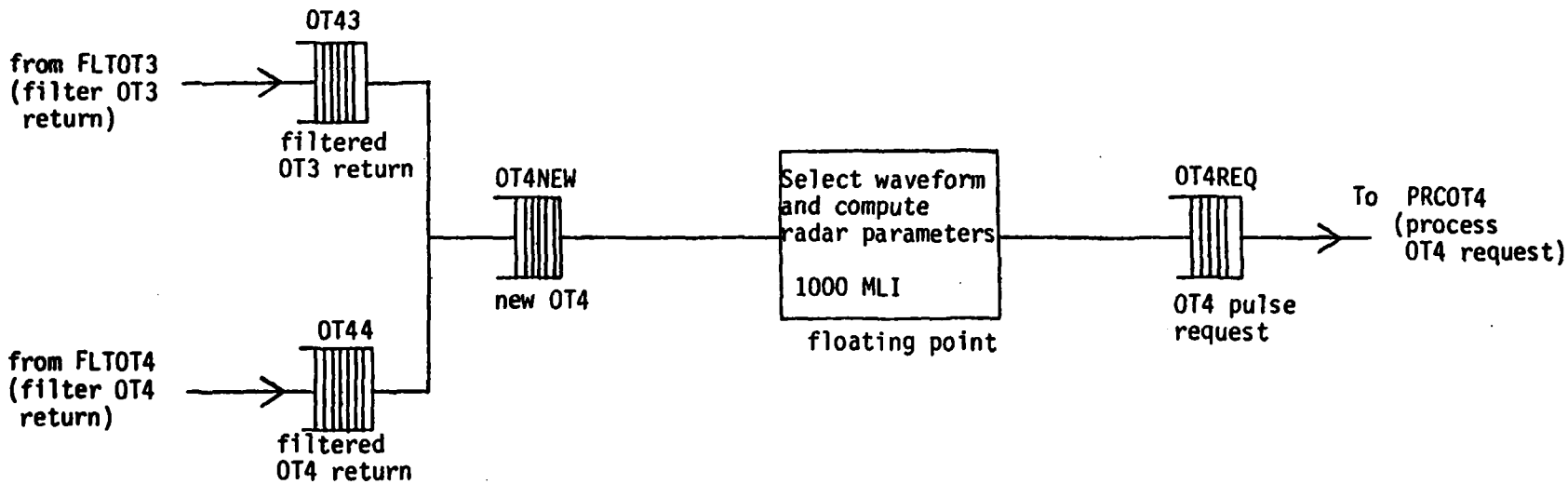
OT returns processing function



Primitive: OT4PLS

Prepare OT4 pulse request

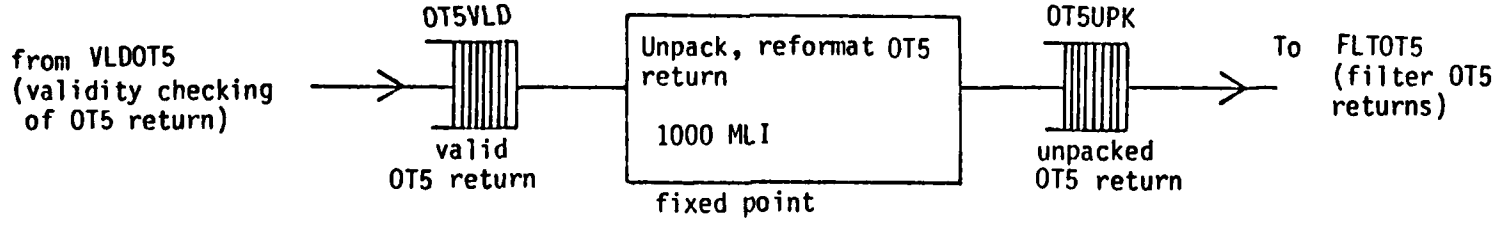
OT returns processing function



Primitive: UPKOT5

Unpack OT5 returns

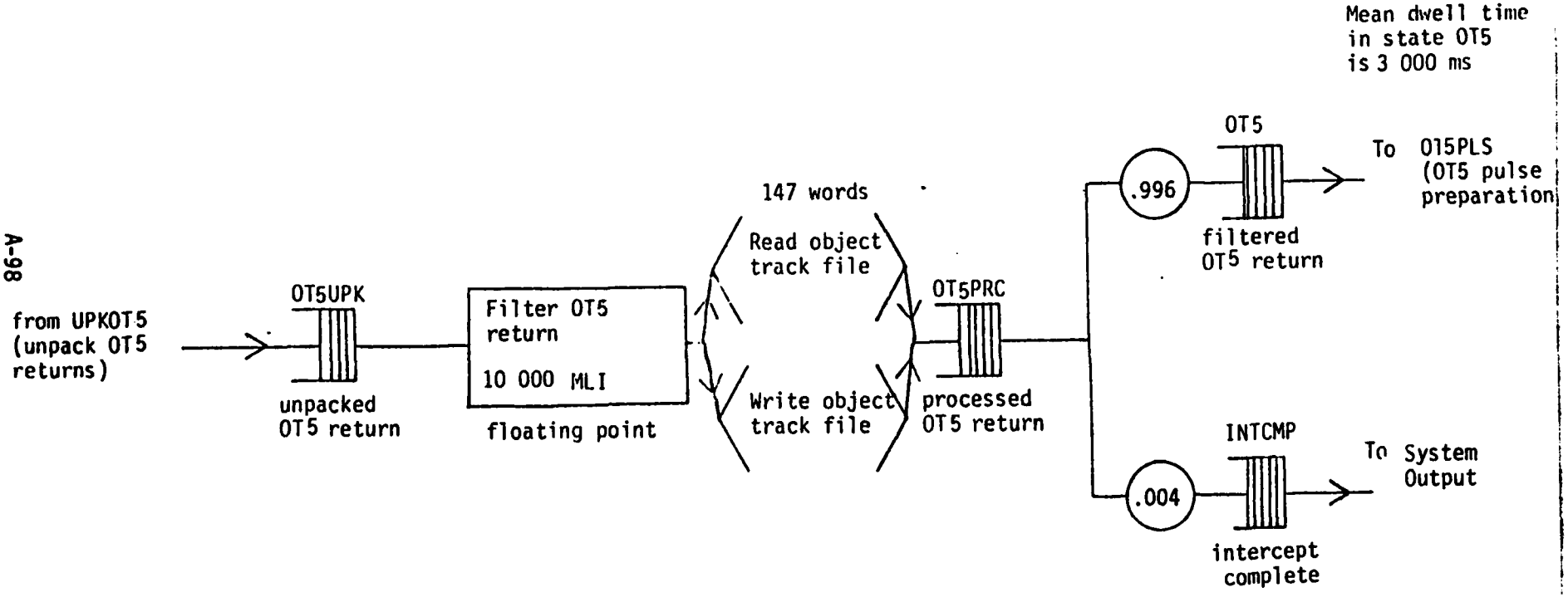
OT returns processing function



Primitive: FLTOT5

Filter OT5 returns

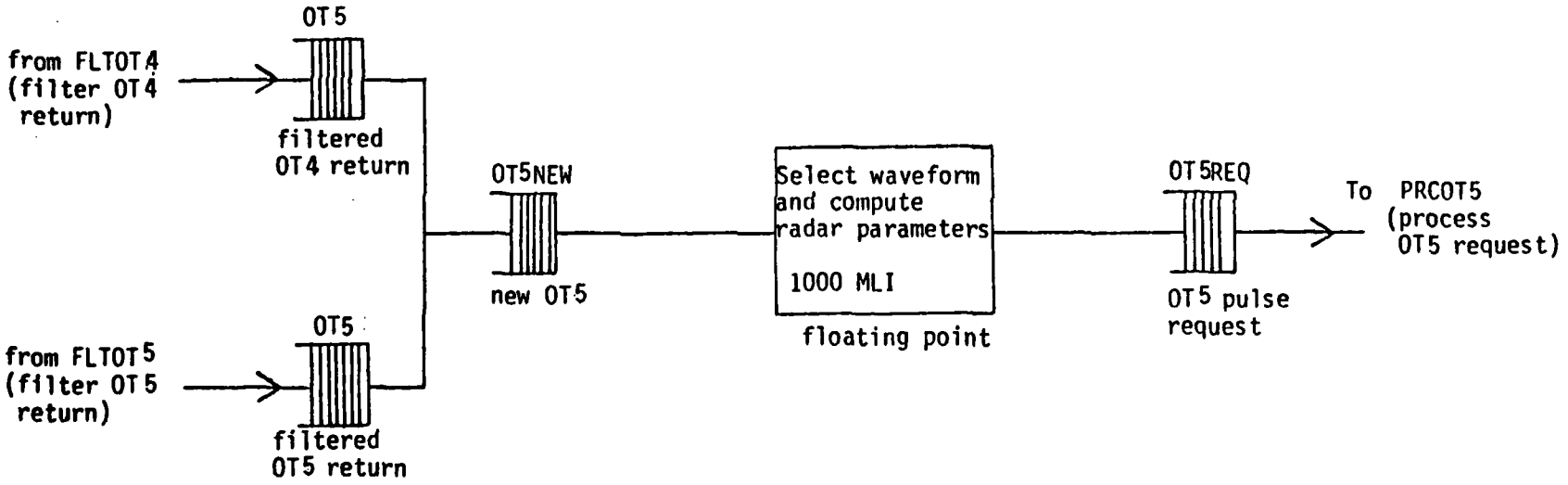
OT returns processing function

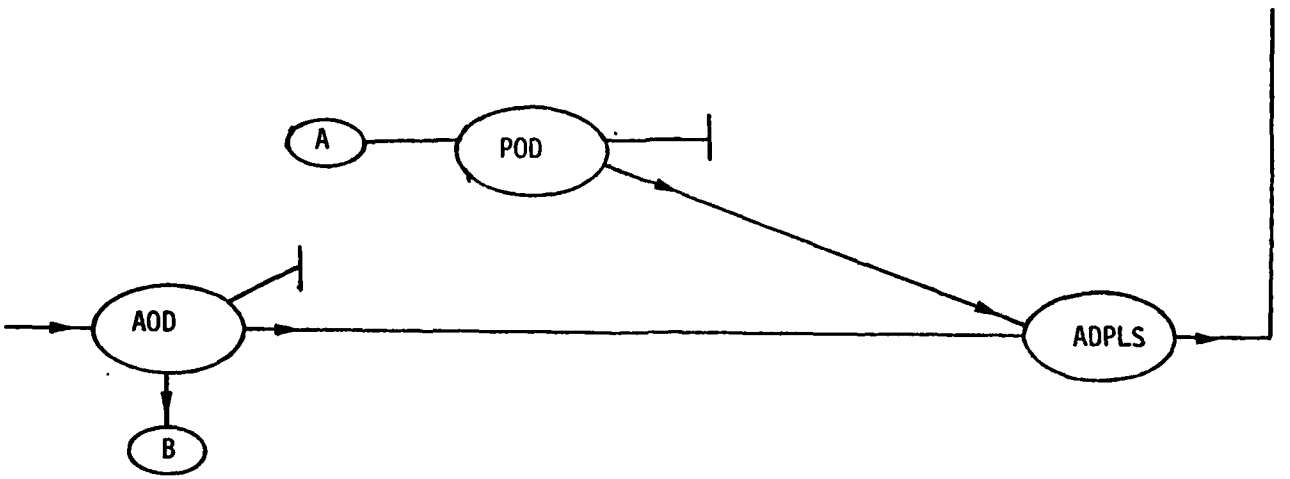


Primitive: OT5PLS

Prepare OT5 pulse request

OT returns processing function



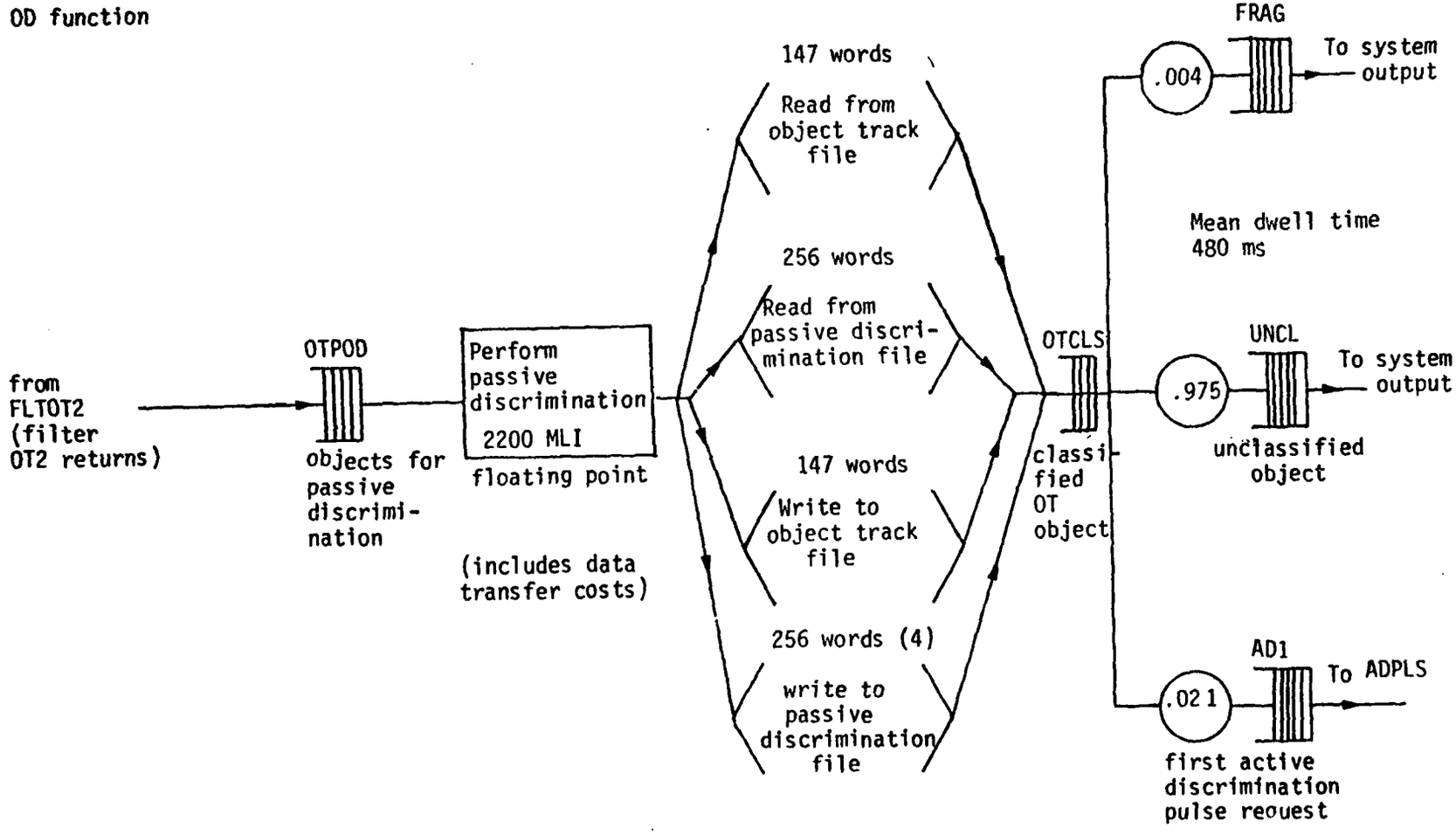


OBJECT DISCRIMINATION FUNCTION

Primitive: POD

Passive object discrimination

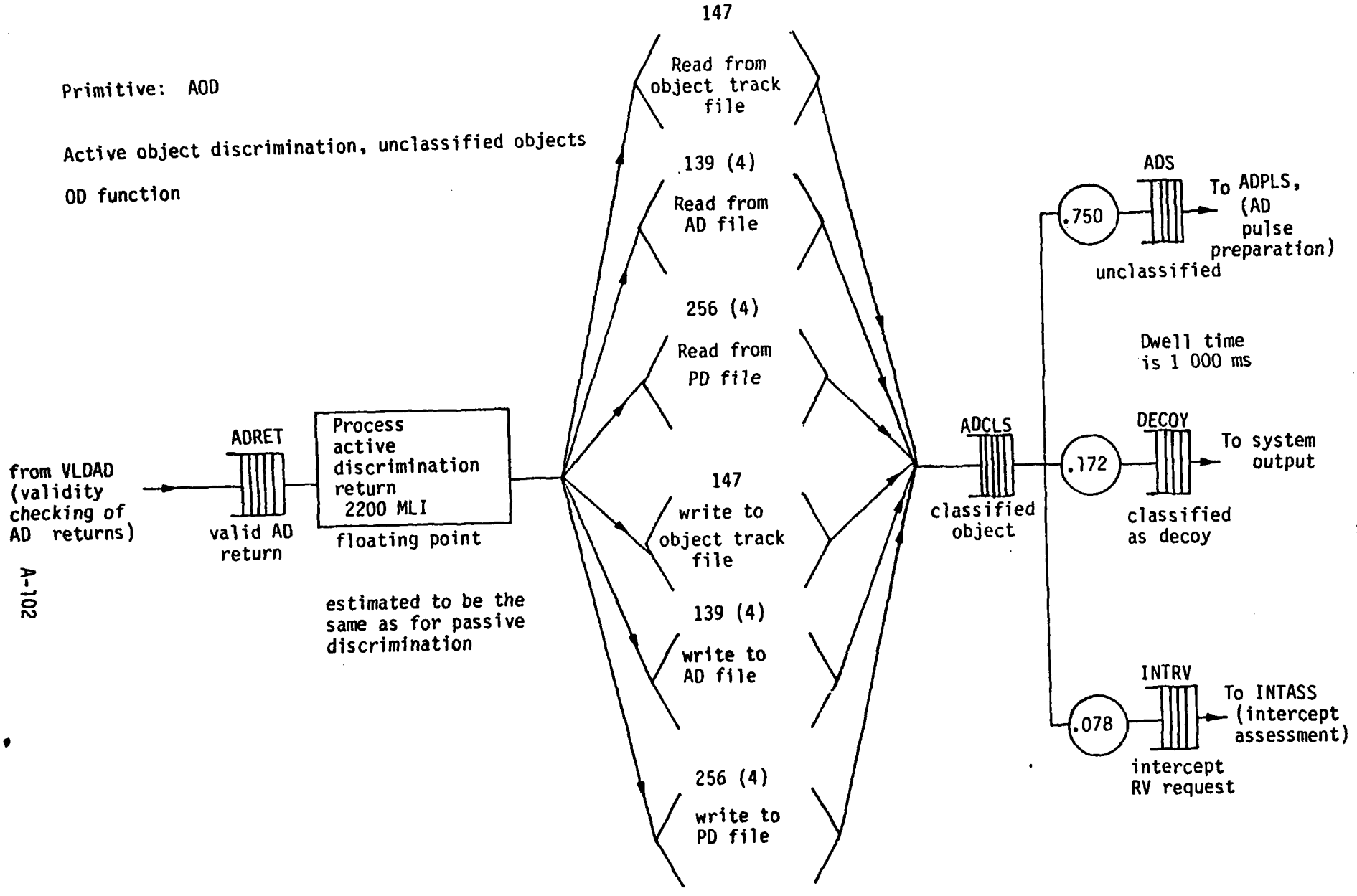
OD function



Primitive: AOD

Active object discrimination, unclassified objects

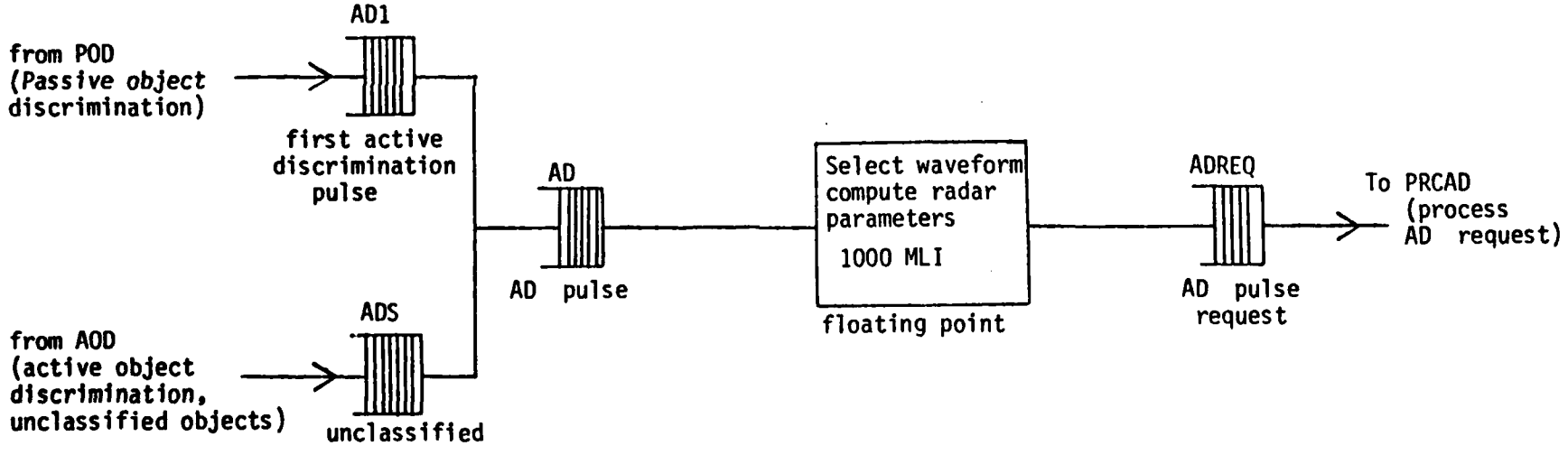
OD function

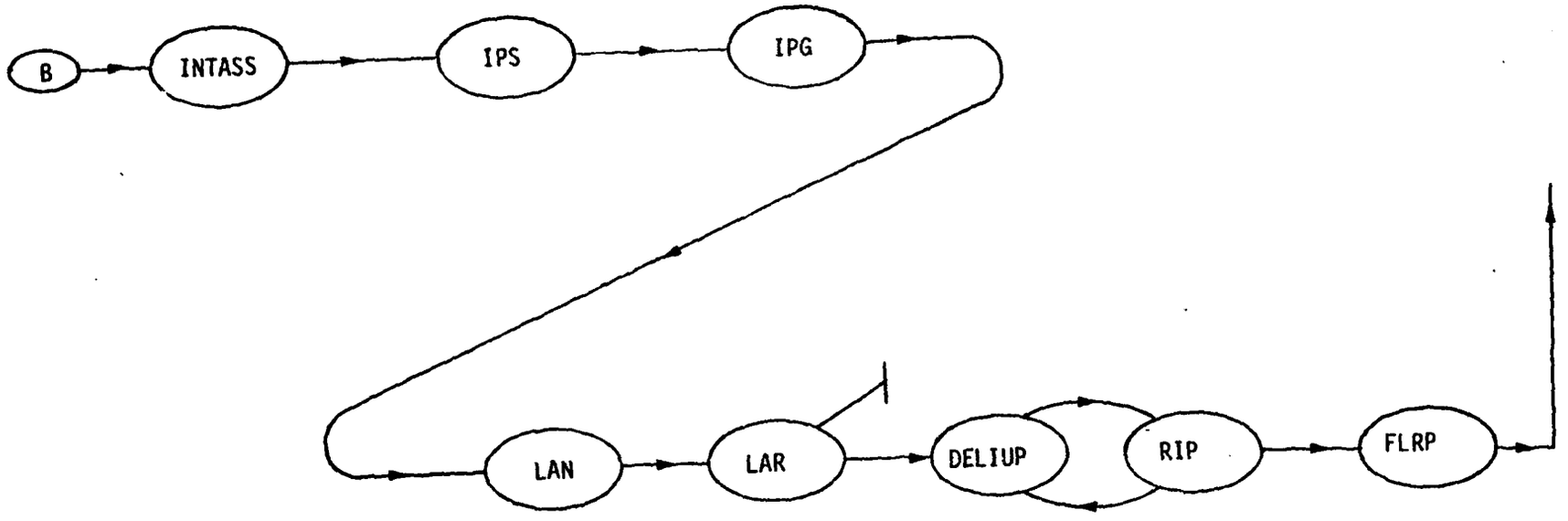


Primitive: ADPLS

First active discrimination pulse preparation

OD function



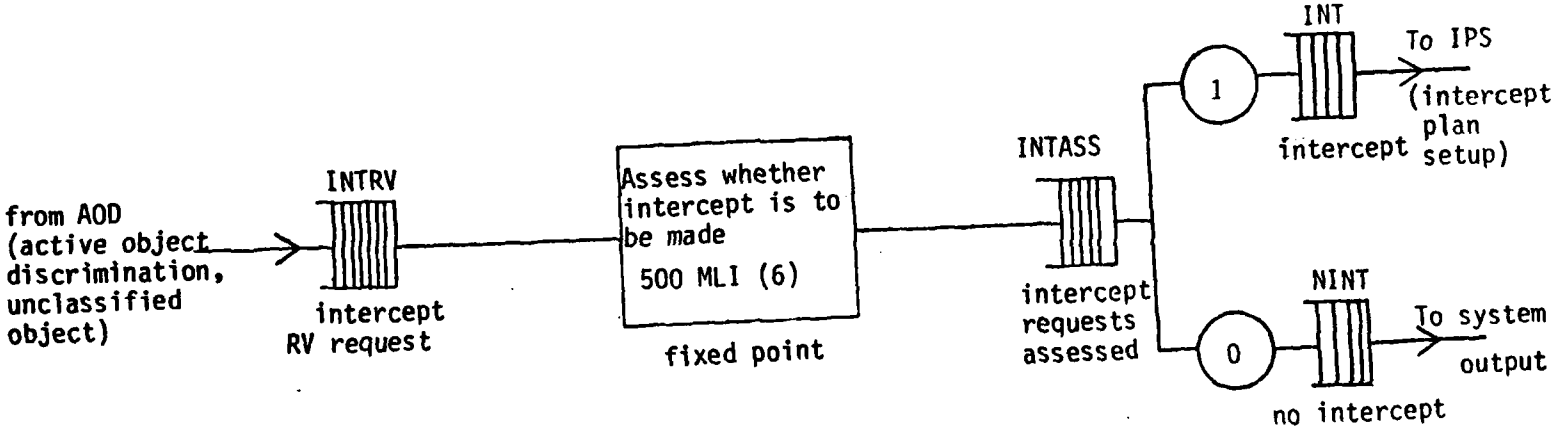


INTERCEPT PLANNING FUNCTION

Primitive: INTASS

Intercept assessment

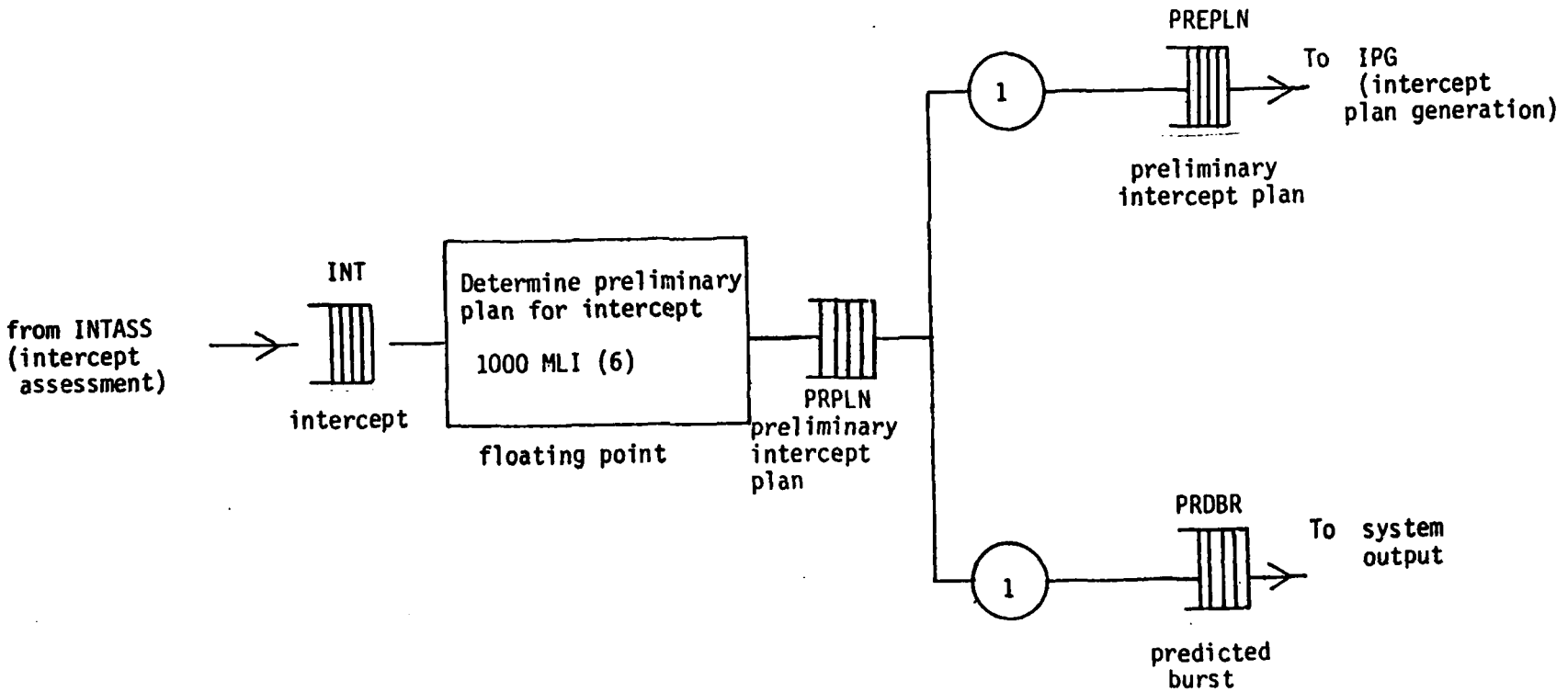
Intercept planning function



Primitive: IPS

Intercept plan setup

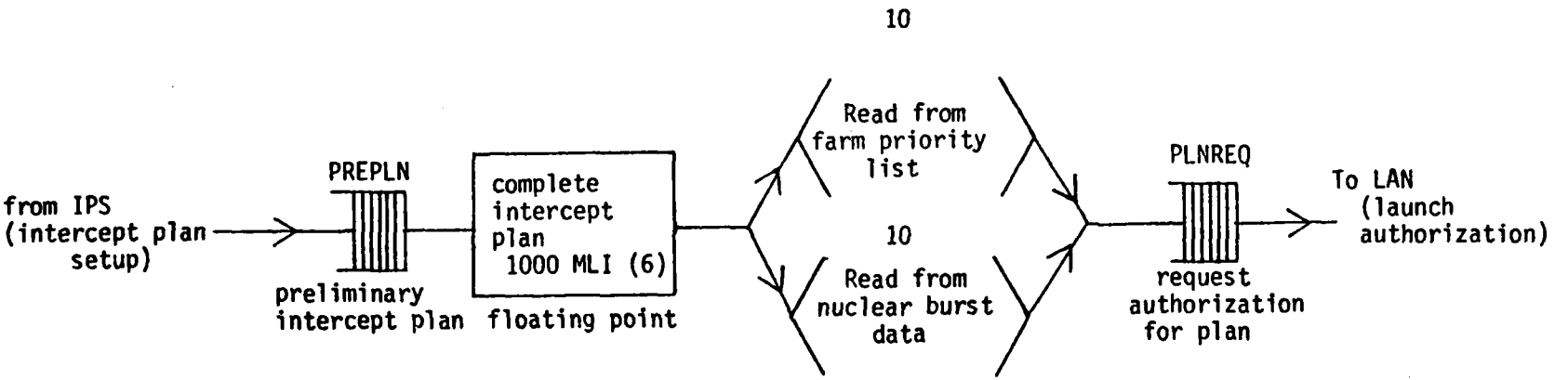
Intercept planning function



Primitive: IPG

Intercept plan generation

Intercept planning function

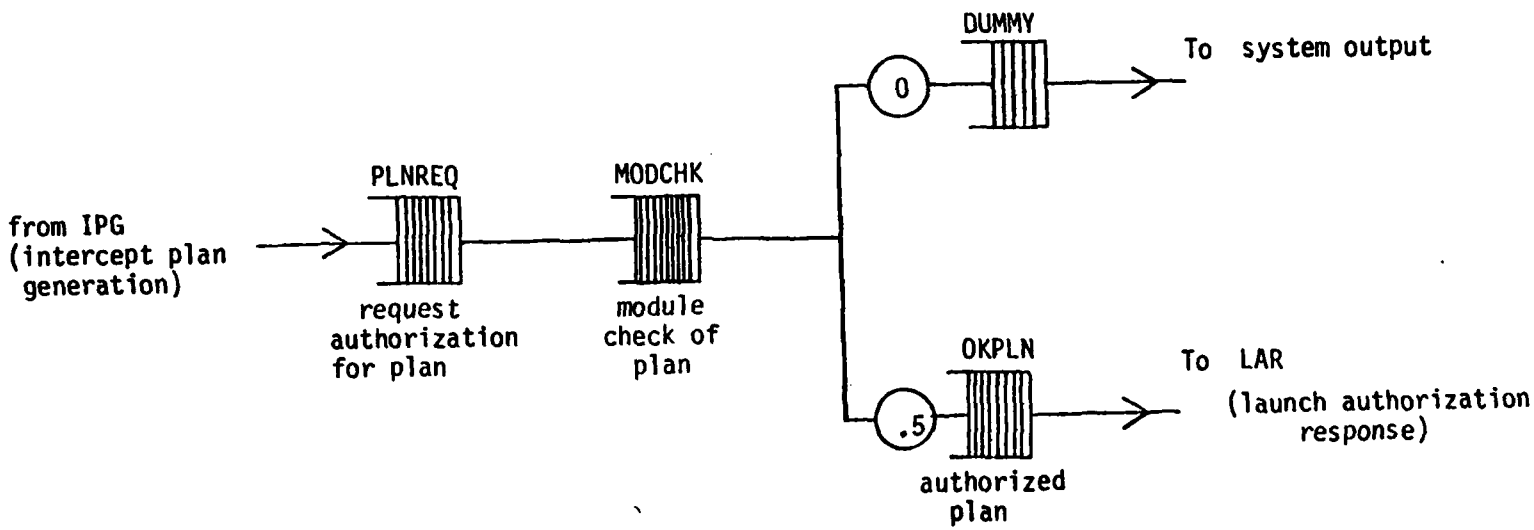


Primitive: LAN

Launch authorization

Module function

Zero-delay

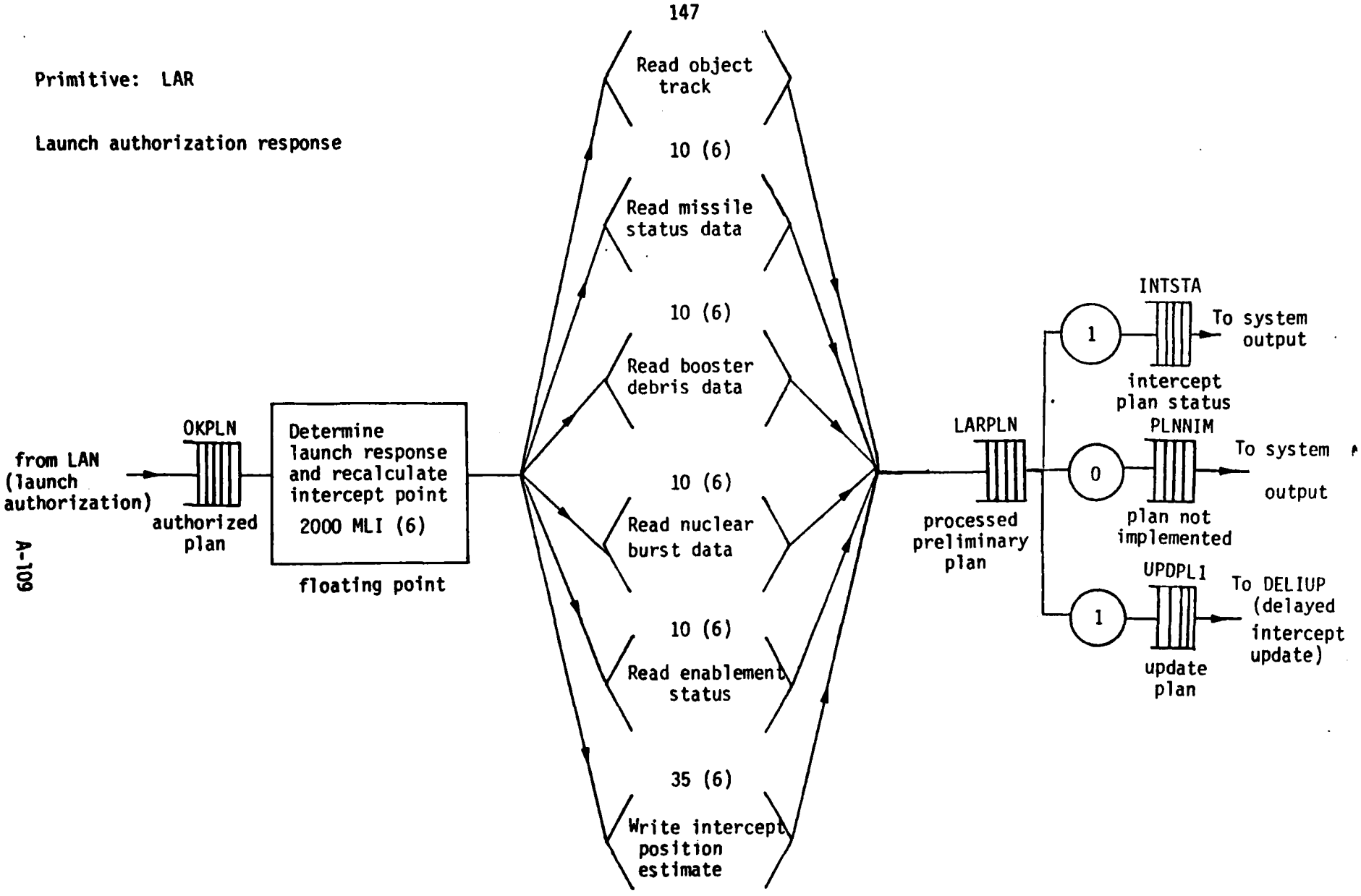


All plan requests are
assumed to be authorized

intercept planning interval take to be a 3 sec delay
.5 of RV's are to be intercepted.

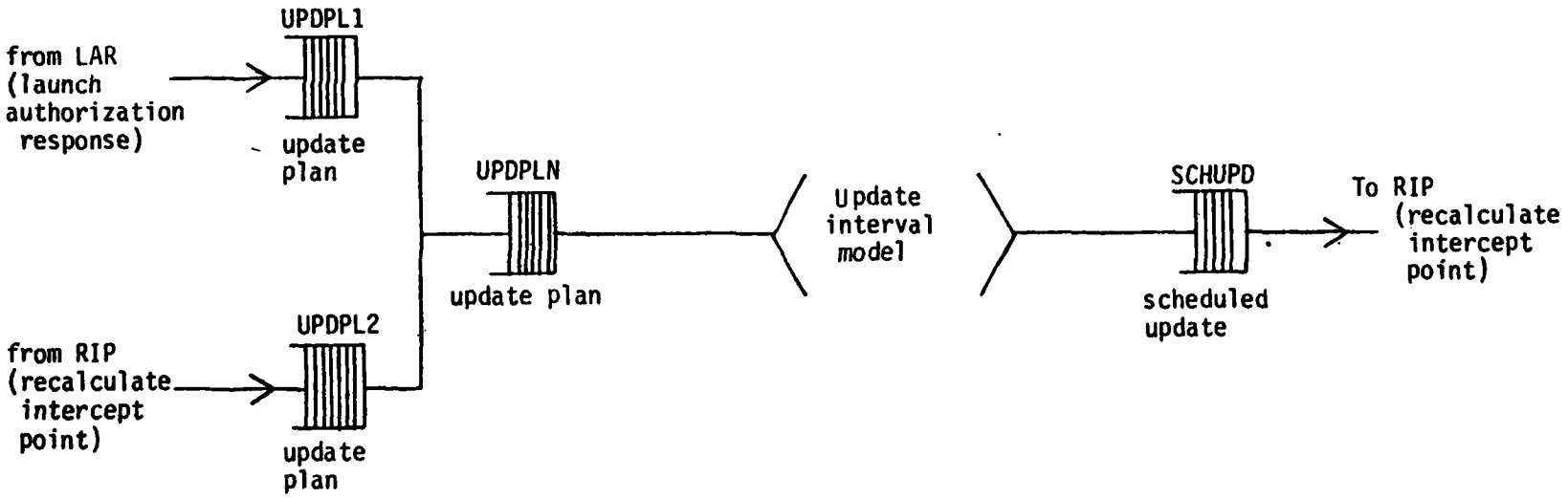
Primitive: LAR

Launch authorization response



Primitive: DELIUP

Delayed intercept update



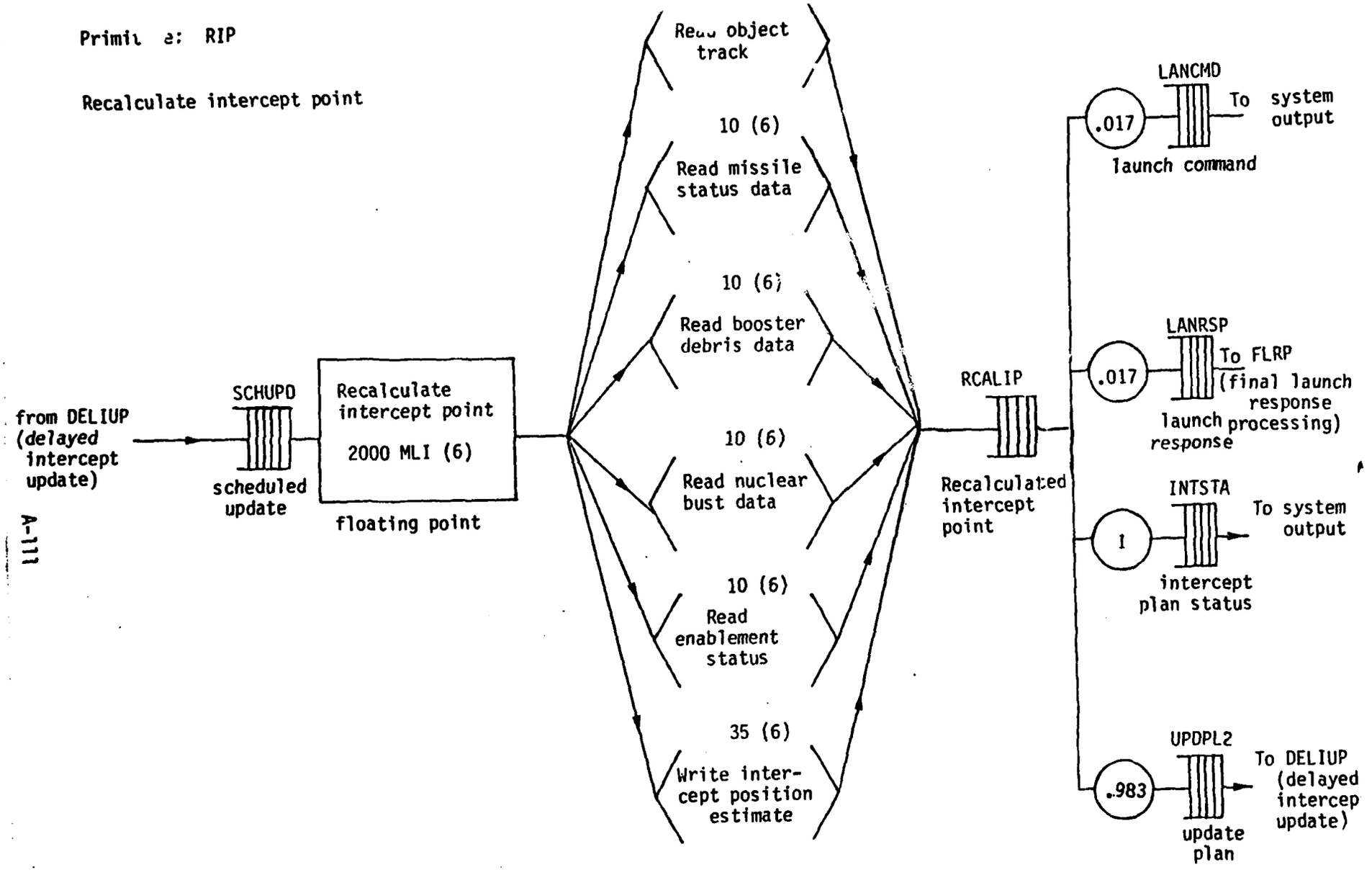
Model for update interval is a pipeline model:

$$\text{rate} = \frac{\text{\# in UPDPLN queue}}{\text{update interval}}$$

Nominal update interval is 50 ms

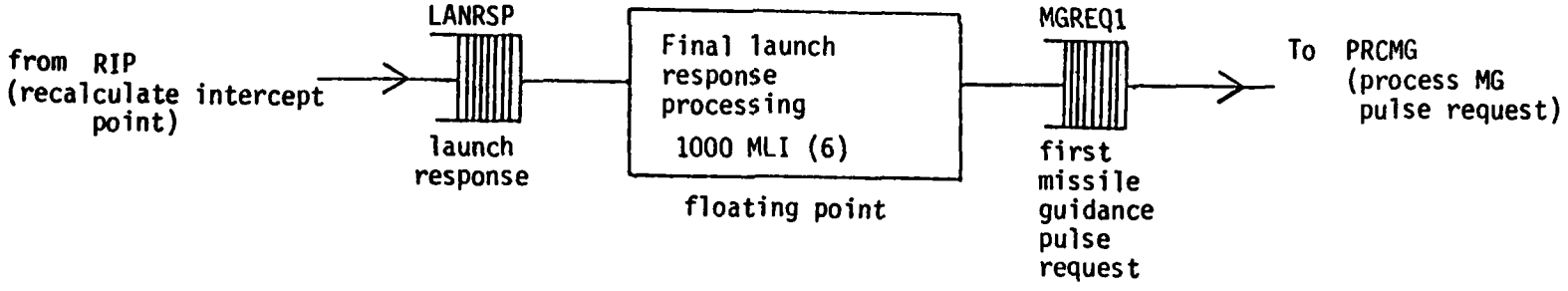
Primitive: RIP

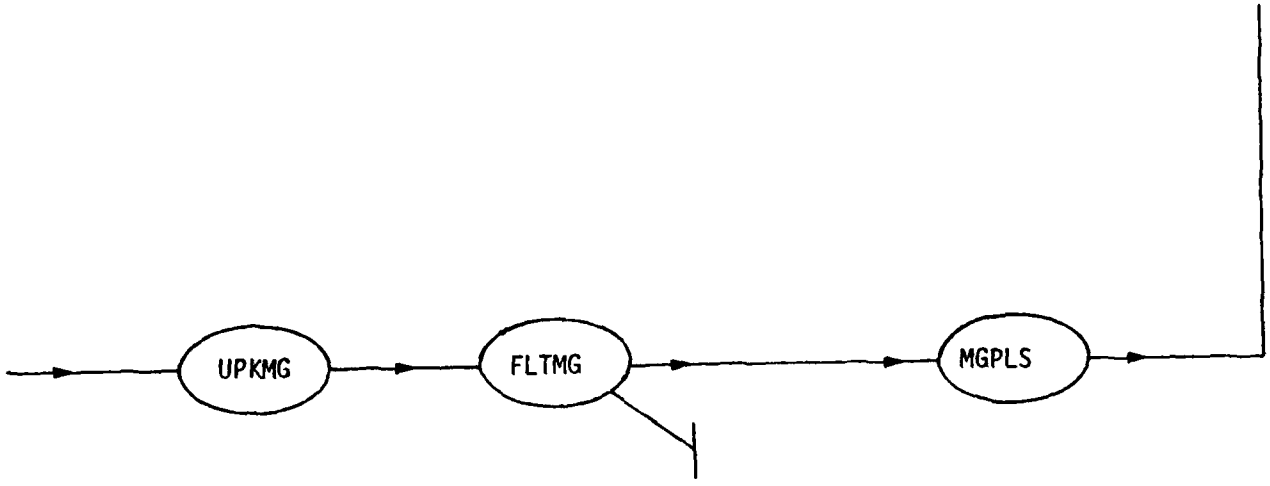
Recalculate intercept point



Primitive: FLRP

Final launch response processing

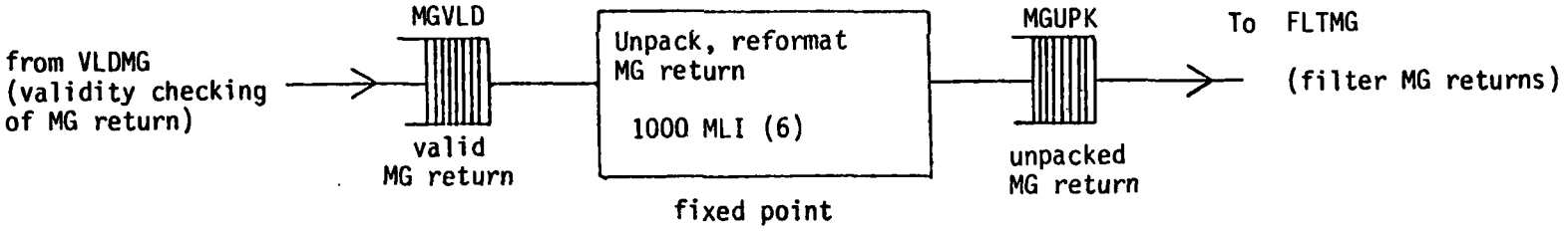




INTERCEPTOR CONTROL FUNCTION

Primitive: UPKMG

Unpack MG return

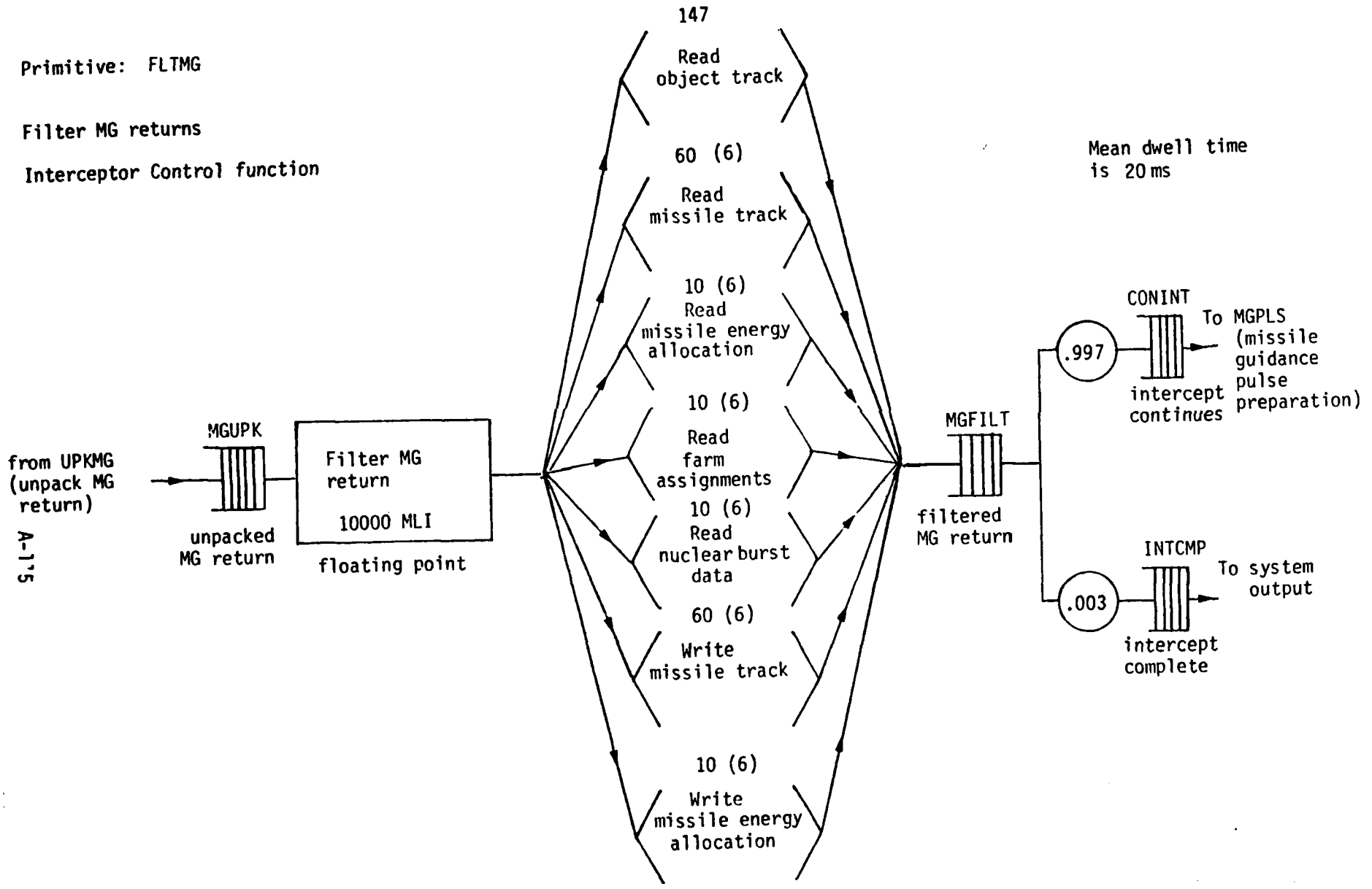


Primitive: FLTMG

Filter MG returns

Interceptor Control function

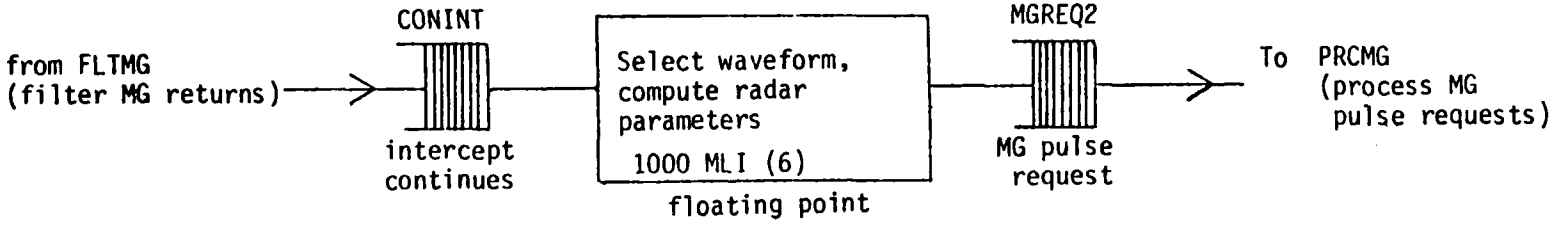
Mean dwell time is 20 ms



Primitive: MGPLS

MG pulse preparation

Interceptor control function



Appendix B

PAES: Processing Architecture
Evaluation Simulation
Approach and Capabilities

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

This memorandum provides a user-oriented description of the Processing Architecture Evaluation Simulation (PAES) that has been developed by VERAC for use in:

(1) Evaluating performance of distributed and central processing systems;

(2) Evaluating and trading-off the performance of communications network protocols and architectures;

(3) Trade-off analysis of processor architectures in performing a specific job, task or function; This includes evaluation of the trade-off between a large central processor and distributed data processing architectures.

1.1 Approach

The PAES simulation simulates throughput, response time, port-to-port delay and other performance factors for a single or multicomputer processing system. The simulation is structured to allow independent specification of the functional process to be realized, and of the computer architecture along with interconnect structure that is to be evaluated. The function to be performed is defined by: (1) a set of elementary functions designated as PRIMITIVES and, (2) a set of LINKS that interconnect PRIMITIVES. PRIMITIVES model an elemental function by means of a set of queues separated by transform elements. The queues contain "work units" and the transform elements cause the transfer of work units between queues in accordance with the rules of an algorithm derived to model the function. The architecture of a computer system is modeled by its effect on transform speed or delay. The performance of an architecture is analyzed by evaluating the number of work units in the queues as a function of time, and by the flow of work units through the functional process.

1.2 Software Overview

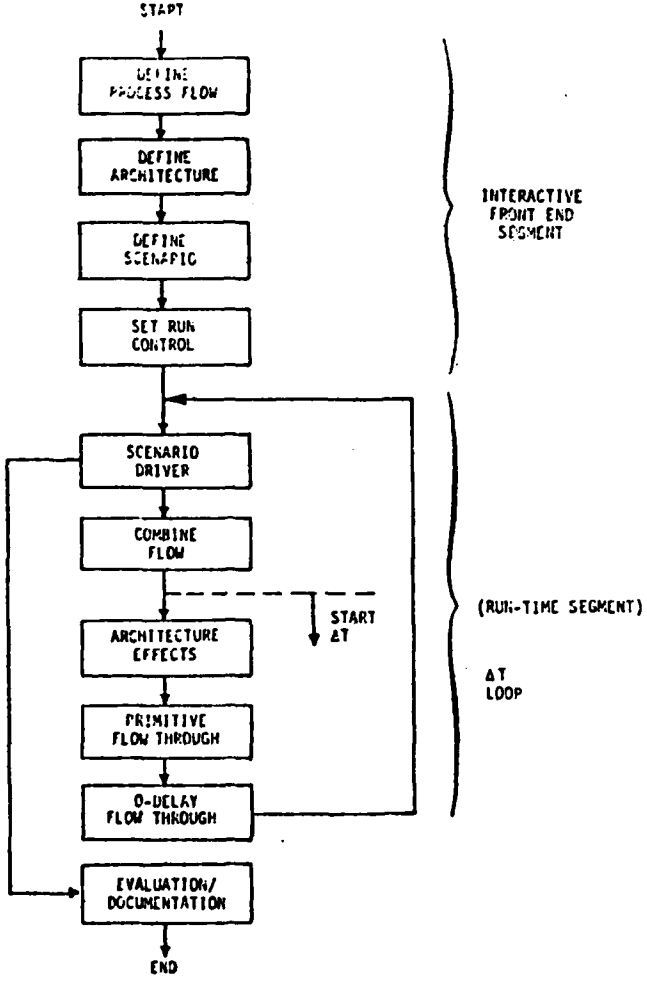
PAES has been designed from the top. The structure is such that the simulation can run on a small minicomputer, but can be elaborated to provide more detailed modeling when resident in a larger computer. This is accomplished by having the models for operating systems, bus control protocols, and external resources as subroutines at the bottom of the code structure. PAES is initially resident on the PDP-11/34 and an associated large disk. The software is segmented with an interactive front-end portion for definition of the simulation data base and a tightly coded run-time portion for the actual simulation run-segment portion (Figure 1). The run-time segment with its data set resides totally in one page (32K/16b) of the PDP-11/34.

The simulation is presently specified to model a process on the order of 150 functional elements (PRIMITIVES). Processor architecture modeling is presently sized at 30 computers with operating systems; 30 intercomputer busses; and 50 resources (disks, terminals, associative memories, etc.). This sizing is oriented to the PDP-11/34 size and speed constraints.

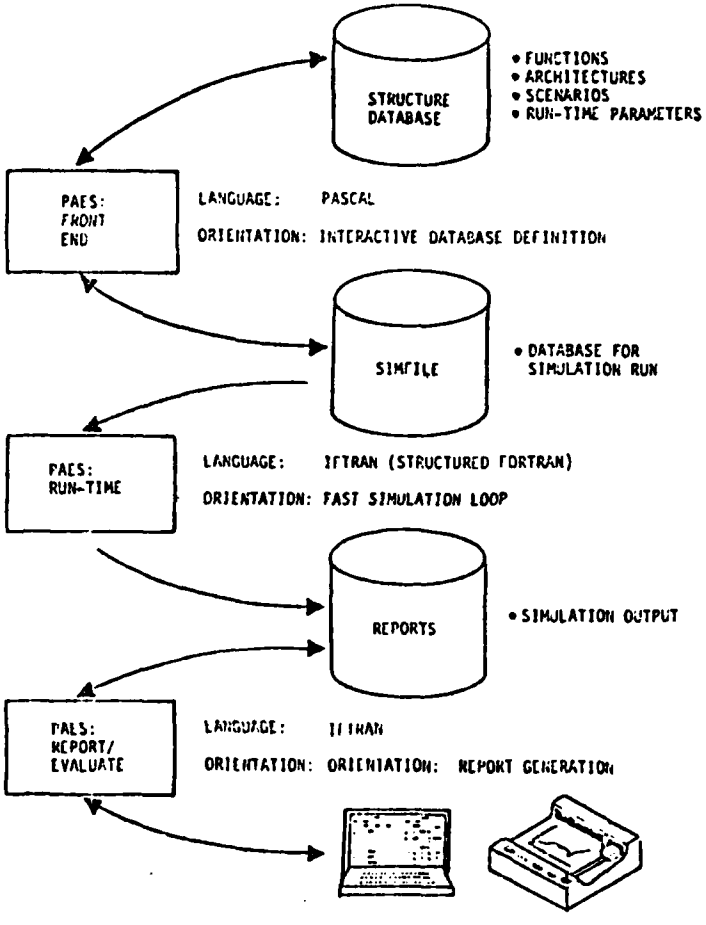
1.3 Verification

PAES provides simulation definition by means of a set of databases that are formed interactively in a series of steps. Each of the databases in the sequence FPRO, ARCH, SCEN, and RTSU relates to the structure imposed by the databases appearing earlier in the sequence.

PAES provides an evaluation of the internal consistency of each database, as well as of the consistency of the database with the set of databases to which it relates. A consistency report is given to the operator when he exits the database definition session.



PAES FLOW PLAN



PAES IMPLEMENTATION

Figure 1. Structure of PAES

2.0 SOFTWARE STRUCTURE

The software is segmented and structured as shown in Figure 1. This structure consists of an interactive portion ("FRONT-END") dedicated to interactive definition of the data bases and implemented in PASCAL, a portion for the actual simulation ("RUN-TIME") dedicated to rapid execution of the simulation and implemented in a structured FORTRAN, and an interactive portion for output formatting and selection ("REPORT/EVALUATE") also implemented in structured FORTRAN. The PAES blocks presented in the top-level flow diagram of the figure are summarized in the following subsections.

2.1 Define Process Flow Block

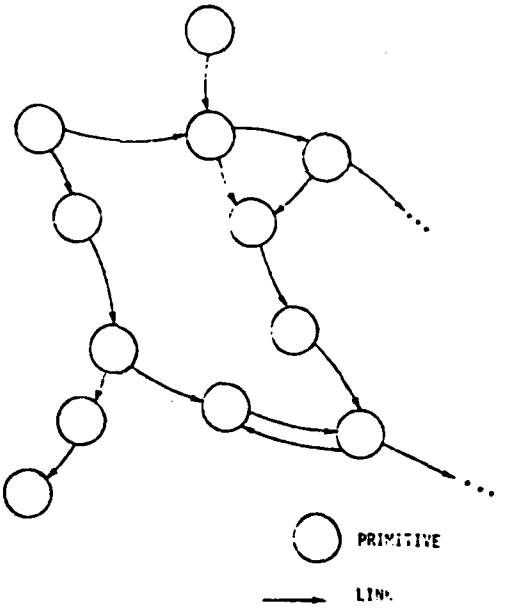


Figure 2. Functional Process Model

The functional process consists of a set of PRIMITIVES and LINKS as depicted above. The task that is to be performed by the processing

architecture which is to be simulated must be defined in terms of these PRIMITIVES and LINKS by the PAES user. The Define Process Flow software block provides the means for the user to build the data base FPRO that contains the functional process definition.

Figure 3 presents an example PRIMITIVE. A PRIMITIVE consists of a sequence of queue sets separated by transforms of various types. Each queue contains "work units". These model meaningful aggregates of data associated with transfer of control (i.e. tokens) that are processed in performing the function. The PAES user must define the work unit at each queue in terms that are meaningful in the context of the function being performed. The PRIMITIVE is structured for:

- multiple inputs
- algorithm complexity modeling
- modeling of references to external resources as well as scheduling constraints
- distribution of outputs.

External Accesses are named, but not related to architecture (e.g. memory devices, terminals, modems, etc.) during functional definition. Algorithm size is modeled in terms of an arbitrary or canonical step size appropriate to the function. Instruction count is not directly specified.

In a later block, architectural effects are modeled as imposing algorithm execution rates and external access delays on the PRIMITIVE. These parameters regulate the work unit flow through the PRIMITIVE

A LINK is structured simply as an input and an output queue that are connected by a path allowing work unit flow. This is depicted in Figure 4.

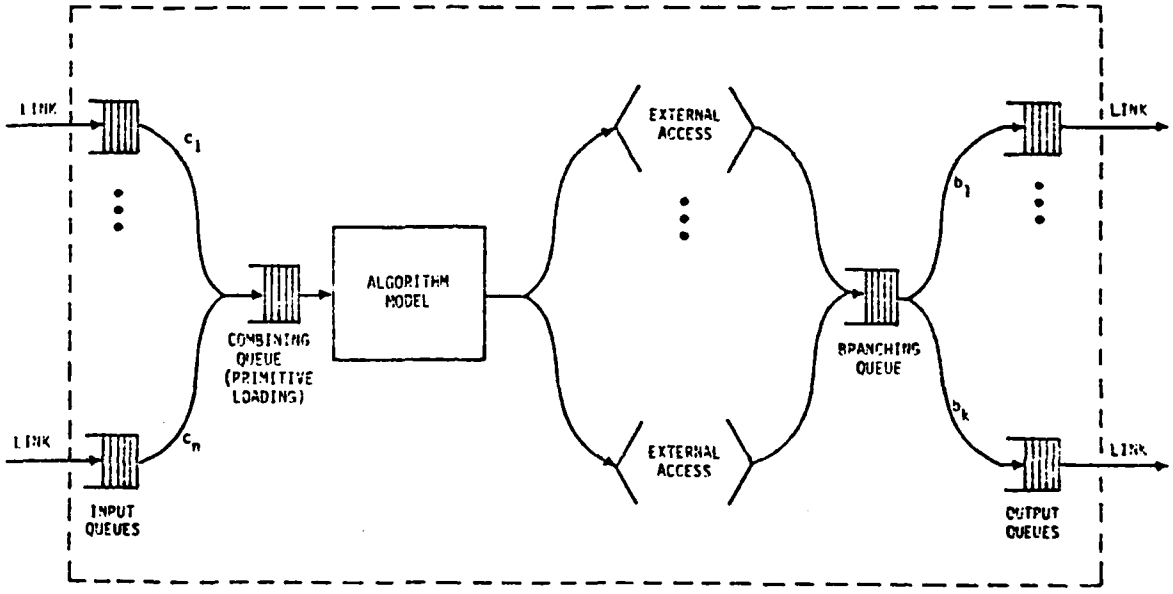


Figure 3. PRIMITIVE Structure

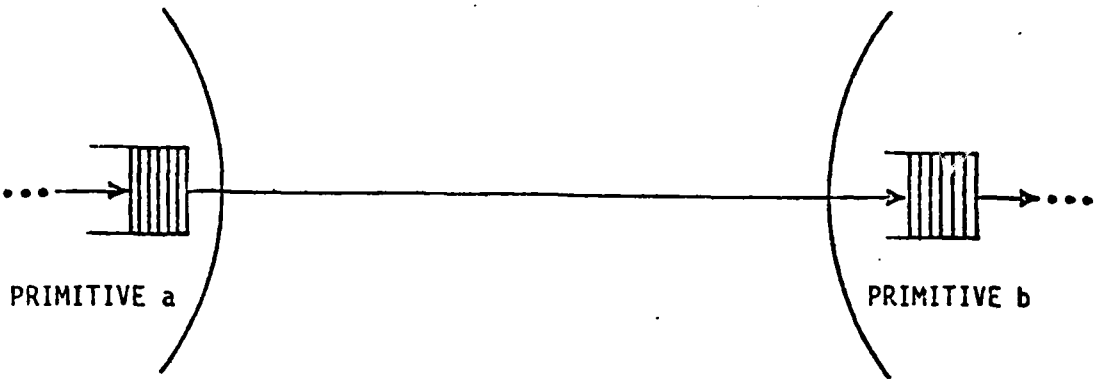


Figure 4. LINK Structure

The LINK queues are specified as part of the PRIMITIVES In which they reside. At the functional level, LINKS contain no structure. They simply model the flow of control between the elementary functional sections (PRIMITIVES).

At an architectural level LINKS can be used to represent activation of subprocesses through an applications operating system; transfer of control within a single program; or transfer of control between processors by means of a bus.

A data size (in bits) is associated with a work unit in the input queue to a LINK. Thus, transfer on data busses of data that is coincident or intimately associated with transfer of control (i.e. parameter set transfer) is modeled within the flow of control context. Transfer of data that is not associated with immediate transfer of control (i.e. data base access) is modeled within the PRIMITIVE as an EXTERNAL ACCESS or, possibly, as an additional number of algorithm steps. The modeling approach selected for this aspect depends on the nature of the data and the designers interest.

This technique for modeling the functional process has two salient, but strongly salutory effects on the system analysis and design effort;

(1) The user is forced to define the problem outside of the context of his notion of the machines that will support the

process. This constraint forces the user to one further level of abstraction at the initial step of the analysis/design process (a higher TOP in TOP-DOWN). The benefits of this can be early revelation of design flaws, and also simplification of the design.

(2) By forcing a structured definition of the problem, a much clearer context for discussion is presented to a set of designers. This allows for clearly articulated design issues and leads to well-defined interfaces.

2.2 Define Architecture Block

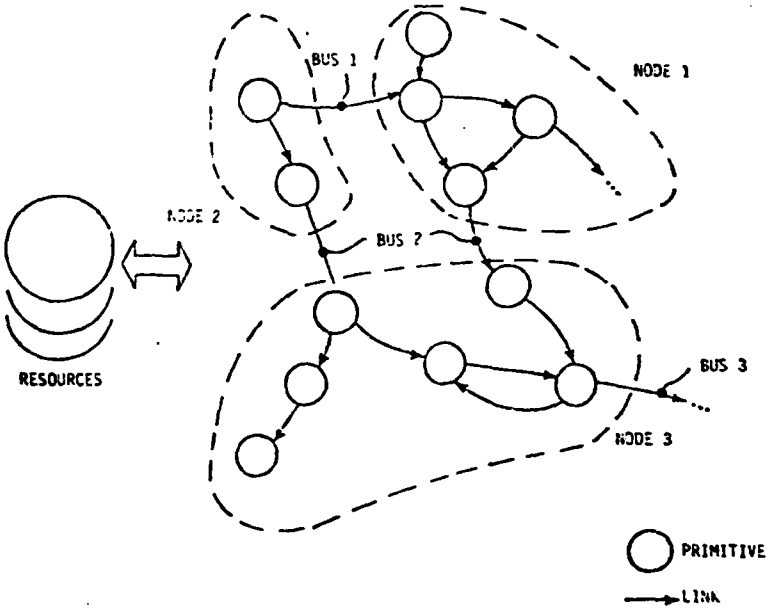


Figure 5. Distributed Processing Architectural Features Imposed Over a Process Flow Model

The processing architecture to be simulated consists of a set of NODES, RESOURCES, and BUSSES. The user interactively specifies the architecture by laying it over the functional process which was defined in FPRO during the Process Definition Block. The structure is depicted in Figure 5. A NODE is a computer in the sense that it contains instructions and a local data memory, and that processing is accomplished under a single operating system model. BUSSES are physical communications channels that carry data between NODES. RESOURCES are devices external to NODES and are accessed for data or for control action in order that functions may be carried out. These accesses are to such devices as shared memories, disks, and user terminals. Process interaction between various work unit flow paths such as scheduling is also modeled as a RESOURCE. The RESOURCE modeled in scheduling is time.

The architecture to be simulated must be defined in terms of these NODES, BUSSES, and RESOURCES by the PAES user. Scenarios to be simulated are represented as sequences of work unit inputs. The scenario input points (External Accesses) are left unassigned during the definition of the architecture.

The three architectural modeling structures are presented in Figures 6, 7, and 8 and are discussed below.

A NODE (Figure 6) is a model of a single processor or computer. The user defines NODES with a certain set of attributes during the architectural definition phase. Basic features defined are:

(1) instruction execution rate, and
(2) an instruction/algorithm-step-size factor for a set of instruction mixes.

The second item allows characterization of the effect of the instruction mix on algorithm processing (e.g. arithmetic fixed, arithmetic floating, string processing, logical, memory access intensive, etc.). Careful modeling of this parameter set can be used to trade-off the effects of

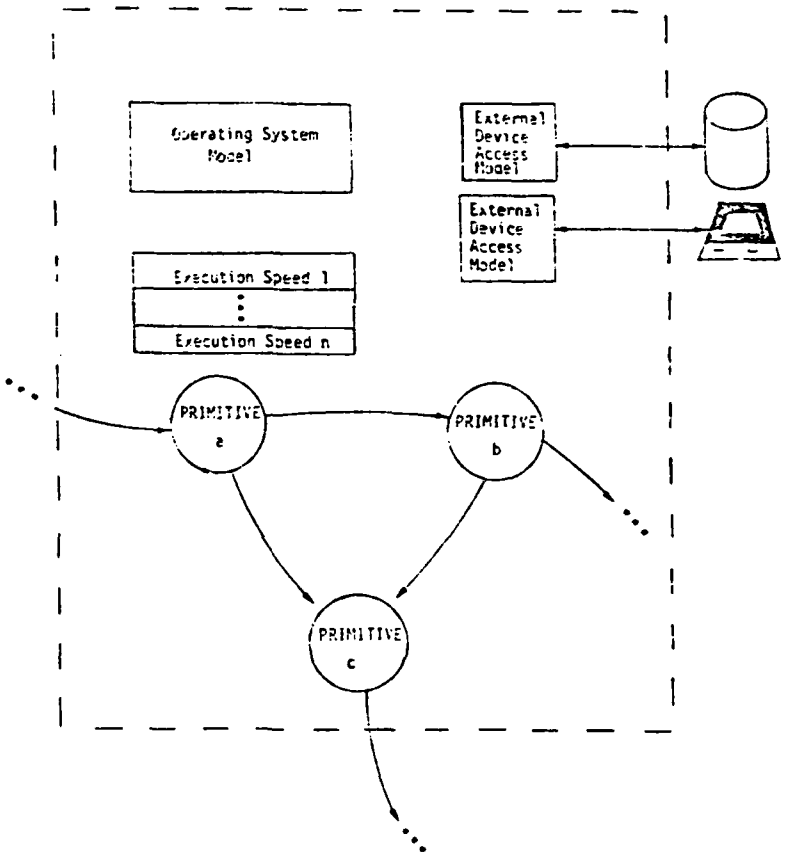


Figure 6. NODE Structure

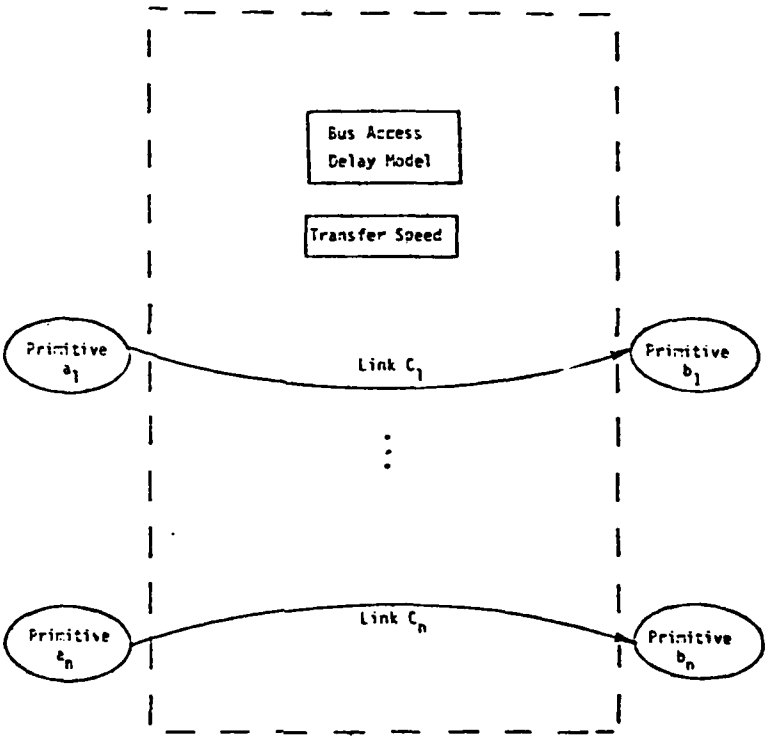


Figure 7. BUS Structure

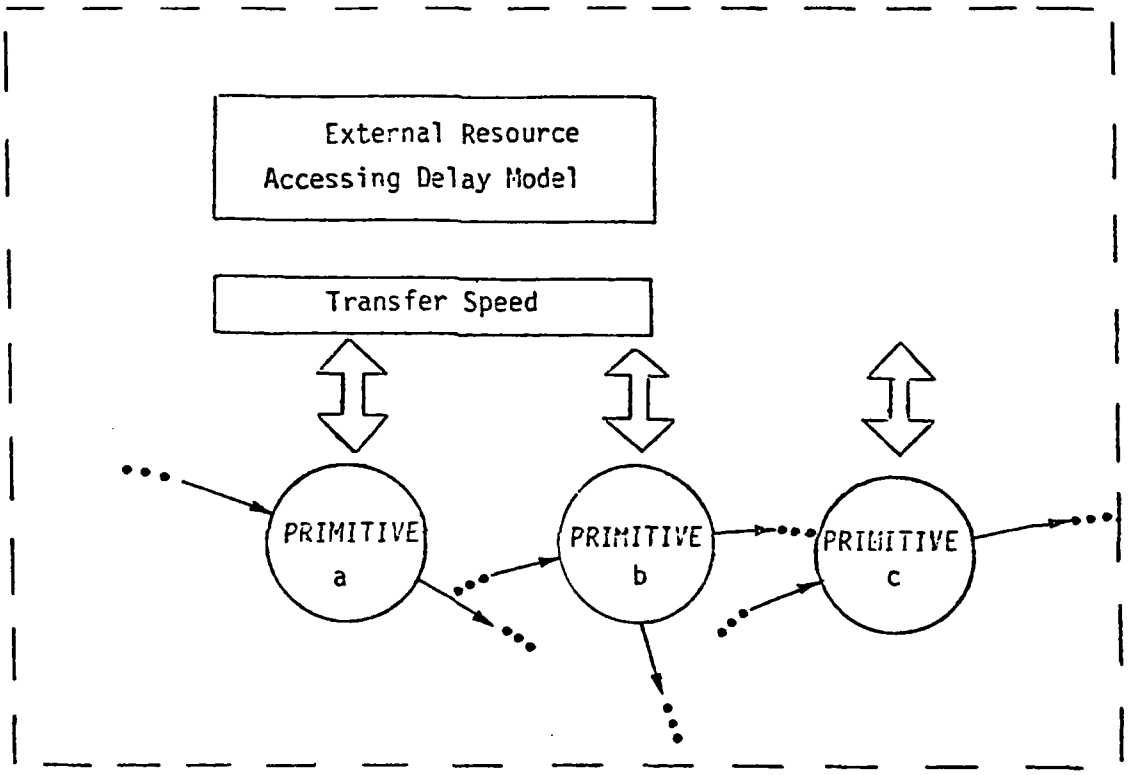


Figure 8. RESOURCE Structure

microprogrammed special-purpose instructions or of hardware features such as hardware floating point or auxillary fast memory.

The tasking of functional responsibilities to computers is accomplished by the operator by assigning PRIMITIVES to NODES. This structure can allow determination of total instruction and data memory required.

An operating system is specified for the NODE. This is mechanized as follows: A set of operating systems are implemented as subroutines at the bottom of the code. New or special purpose systems are simply modeled in a new subroutine. The operating system models determine a proportioning of the computing time to each PRIMITIVE, taking into account the requirement of the operating system itself with its associated handlers. The proportioning is recalculated on each simulation time cycle, ΔT. The operating systems use as input the loading (number of work units) in the Combining Queues (see Figure 3) for the sets of PRIMITIVES that have been assigned to the NODES. The simplest operating system model is a fixed proportioning of the computing time among PRIMITIVES. The most complex model treats a work unit arrival at a PRIMITIVE combining queue as a call to the operating system for a task. This model can include priority queuing, aging algorithms, polling, and variable overhead factors. This model can cause a single PRIMITIVE or set of PRIMITIVES to have time for a number of ΔT cycles representing active tasks in core. Work unit flow in the remaining PRIMITIVES of the NODE can be stopped, thereby modeling inactive processes. Delays for external device data handling in the NODE are also provided by the user to allow this factor to be included in the determination of the performance of a given processing configuration.

The second type of structure included in an architecture model is the BUS (Figure 7). The BUS models physical data channels between processors (NODES). The BUS structure may also be utilized if the designer so selects to model flow of control associated with passing data through a slow memory in a very large computer.

The basic parameter associated with a BUS is the data transfer rate in bits/sec. The BUS model regulates the transfer of work units on LINKS between certain PRIMITIVES. A work unit can be thought of as a packet or block transfer in this context. The size of the work unit (i.e. data block) determines the work unit transfer rate because of loading. Thus, in the LINK input queues, a size, in bits of information, is assigned to work units.

The BUS model is assigned a set of LINKS representing functional data transfer paths that share one physical bus or channel.

Lastly the BUS model is assigned a protocol model selected from a set of protocol models available as subroutines in the simulation. These models, as with the operating system models, reside at the bottom of the PAES code structure. New protocols are easily created or existing protocols modified. The input to a protocol model is the work unit loading of the set of LINK input queues for the BUS. The output of a protocol model is the transfer rates for each LINK in a ΔT period. A simple model might allocate a fixed rate to each LINK (TDMA) independently of the loading. A more complex model might model the delays associated with contention access (ALOHA-type) that are a function of loading. A most complex model would treat the work unit loading as DAMA requests and compute transfer rates based on priority, age-of-request, and BUS control overhead.

The similarity in the underlying structure of the BUS and NODE models is due to the fact that both structures represent hardware resources with an associated resource allocation structure. This same structure is apparent in the modeling of an external resource. The approach of allowing the allocation mechanism to be modeled in as great a detail as the user wishes, but within a very formal structure represents one of the strengths of the PAES approach.

The third type of architectural structure is the RESOURCE (Figure 8). This structure is used to model any delay in flow of control that is caused by a process reference to an external device. Examples of

RESOURCES to be modeled are external memory devices, interactive terminals, and sensors. A more subtle example is scheduling where the RESOURCE model may reference a real-time clock that can be thought of as a "resource". A process factor also to be modeled that is similar to scheduling is algorithmic delay that depends on the state of distinct functional processes. This too can be most conveniently modeled within the RESOURCE structure.

The objective of the RESOURCE model is to compute delays for a set of PRIMITIVES that interact with the RESOURCE. For example, a number of PRIMITIVES situated in distinct computers may require data files from a single disk. The basic parameters to be modeled by the RESOURCE are the data transfer speed and the delay associated with the access (e.g. mean head search time for a disk). These factors are included in the model.

The RESOURCE model is assigned a set of PRIMITIVES and specific EXTERNAL ACCESSES by the user. This assignment reflects the functional role of the RESOURCE. The functional need reflected in the PRIMITIVE EXTERNAL ACCESS such as need for a data base reference is mapped to an architectural structure such as the reference to an external, large core memory or disk.

An accessing model is also included in the resource. This model utilizes the loading of the referencing PRIMITIVES as input and produces access delays for each of the referencing PRIMITIVES. An example of a complex accessing model might be that for a crossbar-switched, multiple-processor/multiple-memory structure such as the S-1 architecture. Access delays associated with the crossbar bus availability could be modeled in great detail in the simulation.

2.3 Define Scenario

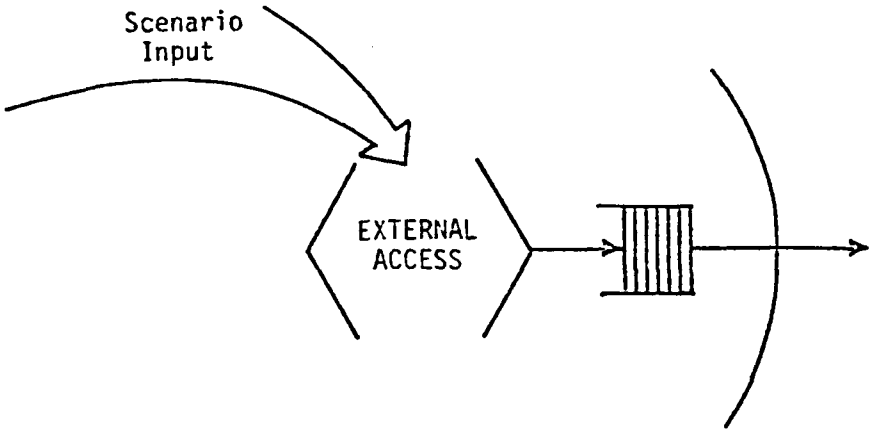


Figure 9. Scenario Input Mechanism

Scenarios are modeled by injecting work units into a set of PRIMITIVES in accordance with a specified time distribution. The scenario is implemented by assigning EXTERNAL ACCESSES to a scenario input structure rather than to a RESOURCE. The user performs this assignment interactively in the Scenario definition block.

Each input stream may be specified by the user as a time function, or it may be specified by the user to reference an external disk or tape file. The first capability allows convenient and expedient modeling appropriate for most analytic requirements. The second capability allows utilization of data recorded during experimentation with the target system environment or with a partially completed system. It also allows creation (on tape or disk) and utilization by the user of scenario loading patterns that are not conveniently represented analytically.

2.4 Run-Time Processes

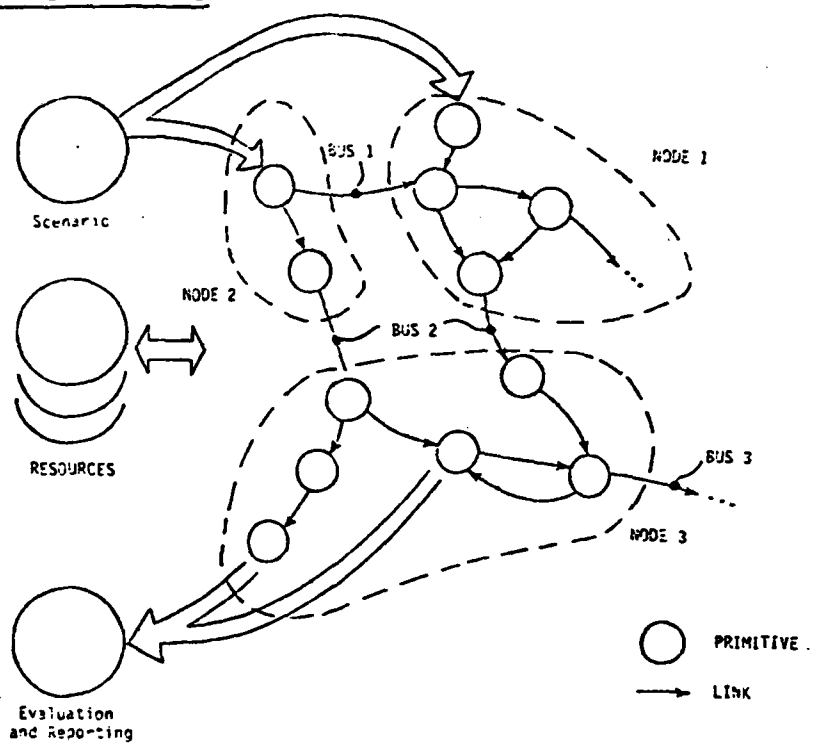


Figure 10. At run-time work unit flow is generated in the functional model. Flow rates are determined by architectural models. Inputs are generated by the Scenario model. Outputs are to evaluation and reporting files.

Run-time processing consists of the following stages (see Figure 1).

(1) Set Run Control - The simulation time increment is set. The total run-time is set. Warning conditions are specified. Output reports and port-to-port tracers are specified. This is performed as the last step in FRONT-END interaction.

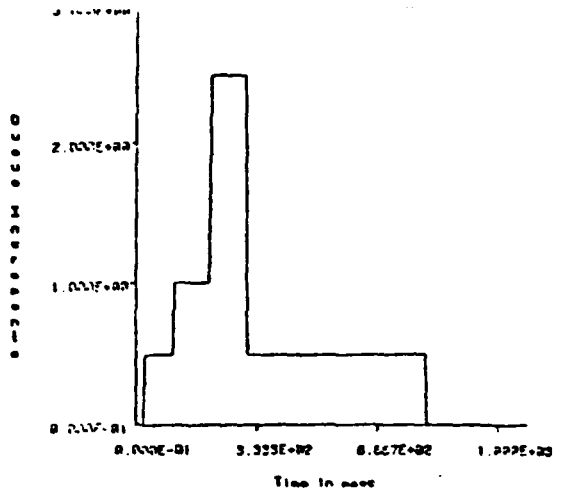
(2) Scenario Driver and Combine Flow - These blocks are located at the top of the actual scenario time-increment loop. They provide bookkeeping functions for the update of the state of the loop in each ΔT increment. These functions include moving tracers within the model and updating the time history of queues being studied.

(3) Architectural Effects - This block provides the computations required for operating systems, bus protocols, and resource allocations. Rates of work unit flow are computed and stored.

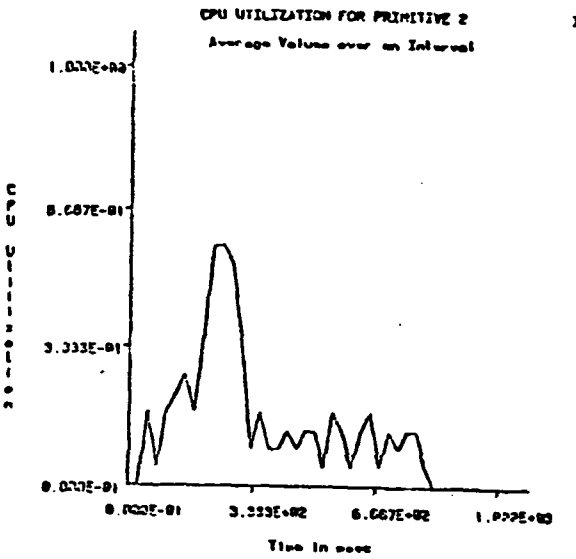
(4) PRIMITIVE Flow Through and 0-Delay Flow Through - These blocks provide the transfer of work units within the model based on the rates provided by the Architectural Effects block.

2.5 Evaluation/Documentation

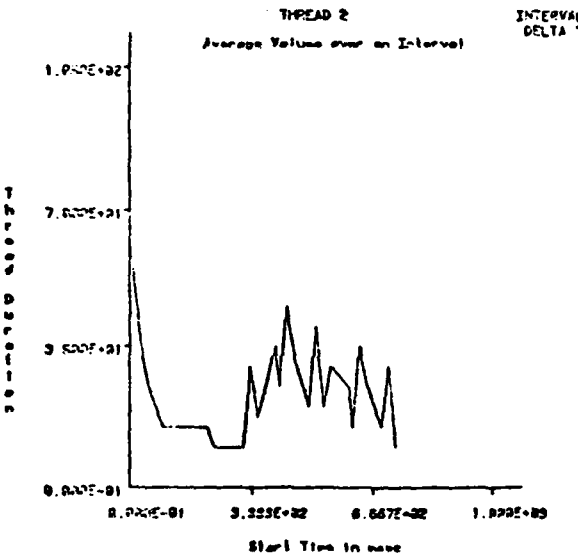
During the run-time of the simulation, files are developed of the loading of specified queues, port-to-port transit times and CPU utilization. The Evaluation/Documentation block allows the user to review these performance curves on a display and to obtain labeled hardcopy graphics of the output, if this is desired. Figure 11 presents example results from a PAES run.



(a) Scenario Input



(b) CPU Utilization



(c) Port-to-Port Response

Figure 11. Example Evaluation Results for the PAES Processor Architecture Evaluation Simulator

3.0 CAPABILITIES AND LIMITATIONS OF PAES

The PAES simulation is time-driven. Thus, it offers a natural vehicle for measuring rates of flow, loading, and utilization for processing networks. The loading of any internal queue, operating system, bus, or resource can be easily observed because of the flexibly structured method of report generation that is embedded in PAES. Time for performance of any function or for sequences of functions related by control flow is provided in PAES by a tracer structure. Figure 11 presents examples of the output of PAES measurements.

The elemental unit of flow in PAES is the work unit. The work units are transferred from queue to queue in the process model at a rate determined by algorithm, access, and architecture modeling. The work unit represents, principally, the flow of control from one process stage to the next. In this sense it is a "token". However, it is the nature of process flow that normally data flows with the transfer of control. It is normal for the "event" of a data transfer to control the "activation" of a subprocess. Work units are given a size in bits to allow modeling of the transfer of data with control. This size can model, as examples, the size of a parameter set associated with an action call by a task to an operating system; the size of a parameter set passed to a sub-block of a program; or the size of a data block transmitted on a bus. Thus, it is natural to model the time-of-arrival of the last fractional increment of a work unit as the time of the action-call to the related operating system or the time of the requirement for databus capacity. This discrete aspect of flow is provided automatically by PAES by allowing only integer work unit transfers across links. Modeling of operating system and external access device interactions with work units is done by the designer.

3.1 Process Design Features

PAES FRONT END flow and database structure is designed to support, in a natural way, the analysis and design of a processing system. A fully interactive interface is supported to facilitate ease of designer interaction with the simulation.

The first stage in the analysis process is definition of the process to be performed. The analyst performs this task in a structured way by developing the PRIMITIVE and LINK structure that defines the problem. When this development occurs within a planned series of reviews and tests with documentation, we consider it a part of a total design methodology. This methodology is denoted PAEM (Processing Architecture Evaluation Methodology). The process or function is defined in database FPRO (Functional Process).

In order for the designer to develop FPRO, he must decide on algorithm sizes, flow of control, database structure, and system input/outputs. The nature of the FPRO model is that the definition of the job to be performed is clearly presented. Also sizing assumptions by the designer are evident. In a formal design environment this structure is documented and reviewed. It is a natural format for ironing out differences in view of the problem to be solved or of sizing assumptions. Specialists in algorithms to be supported can contribute sizing estimates without architectural limitations; separate organizations that are working on different aspects of the problem have a common basis of reference; and users who have specified the work have an opportunity to review a clearly articulated statement of the problem to consider if system requirements have indeed been correctly addressed. These features all promise to lead to solid and consistent problem definition.

Testing of the process definition in FPRO is an extremely valuable stage in design development. Normally, a specification will contain performance requirements consisting of algorithm accuracies, throughput requirements, loading limits, and port-to-port responses. Also, certain architectural features such as a disk memory or a batabus will be specified. Possibly, the processor will be specified and the design problem is limited to structuring the software and operating system. Architectural features can be implemented and specification limits on timing can be implemented as gross delays introduced by pseudonodes. The process can then be simulated at a very high MIP rate. The results

indicate loading throughput, and response times inherent in the specification. Performance inconsistencies that occur at this stage are due to specification inconsistencies only, since no design limitations are imposed. In a PAEM environment a formal review of the process definition and the specification is appropriate at this point.

The second stage of database definition in PAES is the specification of the processing system architecture, ARCH. This includes the set of processors, operating systems, external memories, terminals, modems, and interconnect busses. The ARCH database definition is verified automatically for consistency with the specified FPRO. Multiple architectures can be structured for each FPRO to allow easy trade-off performance analysis.

A computer is modeled as a NODE. A NODE is assigned a set of elemental functions. A NODE is characterized by an instruction cycle rate (e.g., 1 MIP). The designer can select a number of instruction mixes and assign a multiplier to reflect instruction-cycles/instruction for a given mix. The mix type of a given algorithm is specified in the PRIMITIVE. Thus, the designer has great flexibility in modeling the impact of the types of instructions used in an algorithm on the processing speed. This feature can be of immense value when evaluating instruction set options for a microcoded machine and when evaluating the impact of floating point hardware.

The computer model contains an operating system designator and a parameter set. This data is provided by the user interactively to designate the operating system. A set of operating system models is carried in the PAES code as subroutines. The user has the option to provide a new operating system subroutine or to modify an old one. This feature allows the user to select the level of operating system model appropriate for his needs. A very detailed model can be used to study trade-offs in operating system types or in evaluating parameter settings in an operating system. The computer model also carries the list of external accesses occuring from the node. A delay is associated by the user with each access. This delay is to model handler swapping and

processing times. It may be incorporated in the operating system overhead computation.

An interconect between processors is modeled as a BUS. The user defines the bus and assigns functional LINKS to it. The BUS normally models a physical entity that interconnects elements of the system. Examples are a simple RS232 asynchronous line, or a complex Ethernet databus. BUS models can also be used to model transfer of control in large processors where activation of a task involves a significantly slow process of access to a large memory to read activation parameters.

BUS models include a specification of the data rate supported in bits-per-second. The models also include a protocol model implemented as is the operating system for the NODE. The designer interactively designates a parameter set and a protocol type for a bus. Protocol types are carried as subroutines in the PAES code. The designer can provide subroutines for protocols in as much detail as needed. For analysis and trade-off of network architectures, the designer can implement a complete protocol. Note that busses transmit work units and work units model packets of communications data. Thus, the relation between the bus architecture and the real world can be accurately established by a correctly modeled work unit input and a carefully structured protocol.

External resources are the third type of structure defined by the designer. These resources model devices external to the processing chain and complex processes that interact with multiple primitives. Examples of the former are bulk memory devices, modems, and terminals (with operators). Examples of the latter are subprocesses whose processing is a function of the loading on other subprocesses, and secondly, schedulers of work unit flow from multiple primitives.

External resource models include access line rates and a model of access protocol. This protocol model is of the form of the resource control models for the previous two structures - the operating system and the network protocol. The designer can specify a resource type and

a parameter set for each resource modeled. The designer can provide a subroutine to PAES that models an external device to any level of detail of interest. Detailed models would be appropriate, for example, for study of contention resolution approaches for a bulk memory, or, as a second example, for analyzing various scheduling algorithms.

Scenarios are defined by the designer in database SCEN during the third stage of design definition. The designer has complete flexibility in designating the scenario inputs. Scenarios are modeled as work unit loadings on external accesses. The access points are specified by the designer during this stage. Analytic descriptions of loading can be provided by tne designer by interactively specifying a curve type and parameter set for curves carried as subroutines in PAES. Subroutines of arbitrary histograms can also be provided by the designer. The designer can also specify an access to a disk or tape and the access format and record designation. This allows scenario data to be used that is derived from real-world test and experimentation. It also allows use of data generated by other simulations or in a master facility. Thus, uniformity of testing can be guaranteed.

Multiple scenarios can be structured and stored for use with single or multiple architectures. Thus, trade-offs on scenario sensitivity can be easily performed.

Consistency checks of a scenario definition with an architecture are provided automatically by PAES.

The designer specifies the simulation run to be performed in the fourth stage of PAES. This interactive stage allows him to easily modify simulation run characteristics in order to evaluate the simulation performance itself, or to vary the features being examined.

PAES is structured to provide complete transparency of the simulation to the designer. He can specify any of the following three types of reports:

(1) Queue loading as a function of time; (Inflow to PRIMITIVES can also be reported).

(2) CPU, bus or resource loading in MIPS, bits, or accesses can be generated. (Bus and resource loading has not been implemented yet.)

(3) Tracers to measure transit time between any two queues along a specified thread of control flow.

Start times for thread tracers can be specified by initial thread time and interval between times. This allows a complete history to be generated of port-to-port response under scenario loading.

3.2 Simulation Run-Time Features

Timing parameters that characterize the simulation are as follows: The simulation updates the state of the process flow each ΔT seconds. Parameter ΔT is typically 1-10 msecs. A period of 1 msec. to 32 seconds can, at present, be simulated for the total run. Intervals are now set to 40 x ΔT. Intervals are the time-increment at which report blocks are shifted to disk files and scenario histograms are taken in.

Certain PRIMITIVES and LINKS are denoted by the operator as 0-delay. These do not effect work unit flow. Thus, structures such as external accesses and algorithms can be included in the process model, but need not be considered in the simulation run if they are not appropriate for a particular design approach. This is particularly appropriate to memory structure modeling where external memory devices may or may not be used. Thus, the delay associated with external memory access may be zero. A PRIMITIVE or LINK is also set to 0-delay (default) if the delay imposed is typically very small. Warning flags are set if this condition occurs during a simulation run.

The simulation moves work units through one PRIMITIVE and one LINK, at most, each ΔT interval. Thus, nonzero delay primitives introduce a minimum delay of ΔT to each work unit flow.

During each ΔT increment all operating system, bus protocol, and resource access algorithms are updated, giving a fully dynamic model of these processes.

Also, during each ΔT increment reports for loading and response are updated giving a complete view of the evaluation of the process. A special feature allows measure of work-unit-inflow/ΔT-increment as well as loading/ΔT-increment for the PRIMITIVES.

Port-to-port response time measurement is accomplished by tracking a tracer work unit through a thread that has been specified by the operator. Tracers are updated each ΔT increment. The work unit traced is either one that is available in the starting queue (from scenario evolution) at start time or is a pinch of work unit (.01 wu) added by the tracing procedure. Traces can be specified to occur every few increments to create a history of port-to-port response time as illustrated in Figure 11c.

3.3 Report Generation Features

VERAC has emphasized providing ease of designer-access to PAES capabilities. This is reflected in the availability of output data for the designer. Raw data is stored by the simulation on report blocks on disk. The report program accesses and manipulates this data to present graphs of desired performance features. This data is presented to the user on a graphics display terminal. The user can vary the presentation and then call for a plot on a flat-bed plotter. Figure 11 presents example reports.

All warnings generated during the simulation run are presented to the user on-call during the report/evaluation phase.

4.0 SUMMARY

PAES provides a user with a processor architecture evaluation tool that is designed for great flexibility and user convenience.

PAES provides a vehicle for consistent process definition. Process definition is easily structured into a methodology (PAEM) that provides visibility and review to the process definition (i.e. statement of problem). The PAES format of LINK/PRIMITIVE representation with algorithms and accesses clearly denoted, provides a basic reference or context within which related design and evaluation teams can develop a concensus statement of what needs to be accomplished by the system.

PAES provides a method of specification review or generation. Specification limits can be implemented in a pseudo-architecture on the process. Specific architectural elements (processors, memories, etc.) can be included in this model. Simulation runs indicate performance features implicit only in the specification.

Multiple architectural approaches can be easily specified and tested against the basic process. Trade-off and sensitivity analysis are, thus, easily accommodated.

Multiple scenarios can be tested against a given architecture to evaluate sensitivity of the structure to this variation. Scenario input sources can be analytic or histogram models or externally provided data files.

The user can model key features of interest to great detail - at his option. Features amenable to this are: Operating system models, network protocol control models, resource access models, schedulers, and scenario inputs.

All internal queues can be examined by the user. All thread transit times can be examined by the user. All resource loading (operating systems, bus controller*, and resource access controllers*) can be monitored.

The input and output of PAES is directed toward user convenience. Databases are constructed using interactive editors especially structured for each database. Output formats can be generated, viewed on a display, and plotted if desired. Thus, the user can move rapidly through analysis and design of a complex system.

*to be implemented

Appendix C

Detailed Architecture Evaluations

This Appendix provides detailed results of evaluations of data processing subsystem architectures. The architectures evaluated are:

(1) Centralized,
(2) Thread, and
(3) Hybrid.

These architectures are described in detail in Section 4.0 of this report.

Each architecture was evaluated for two scenarios:

(1) single-spike, and
(2) double-spike.

These scenarios are described in Section 3.0.

The process that is executing on these architectures is described in overview form in Section 3.0 and in complete detail in Appendix A.

The simulation tool PAES was used to provide the evaluation. This tool and the methodology PAEM of which use of PAES is a part is described in overview form in Section 2.0 and in further detail in Appendix B.

The evaluation results presented here consist of cpu utilizations for computing "nodes". In the Centralized Architecture, there is a single node, CENTRAL.

In the Thread Architecture, there are nine computational nodes, as follows:

RRA	Radar return assimilation
MACRO	Radar macro scheduling
MICRO	Radar micro scheduling
SV	S/V returns processing

TI	Angle group (Track Initiation) returns processing
OTOD	Object track and discrimination
REDUND	Object correlation
IP	Intercept planning
IC	Interceptor guidance and control

In the Hybrid Architecture, there are four computational nodes, as follows:

RRA	Radar return assimilation
MACRO	Radar macro scheduling
MICRO	Radar micro scheduling
AGGPRO	Aggregate processor, representing the set of THREAD processors performing all other functions

The results presented in the following are organized as follows:

Centralized Architecture

CENTRAL node, single-spike scenario: Figure C-1
CENTRAL node, double-spike secnario: Figure C-2

Thread Architecture

Each of nine nodes, single-spike scenario: Figures C-3 to C-11
Each of nine nodes, double-spike scenario: Figures C-12 to C-20

Hybrid Architecture

AGGPRO node, single-spike scenario: Figure C-21
AGGPRO node, double-spike scenario: Figure C-22

Note that the remaining three Hybrid Architecture nodes - RRA, MACRO and MICRO - coincide with three of the Thread Architecture nodes.

Each figure consists of a pair of plots. The first presents "Sample Values", each of which is the loading measured over the 2 ms integration step interval used in PAES. The second presents "Average Value Over An Interval". These values are averages over a 60 ms interval. The averages represent cpu utilization where a modest queueing of work units is involved, and therefore are more indicative of cpu requirements.

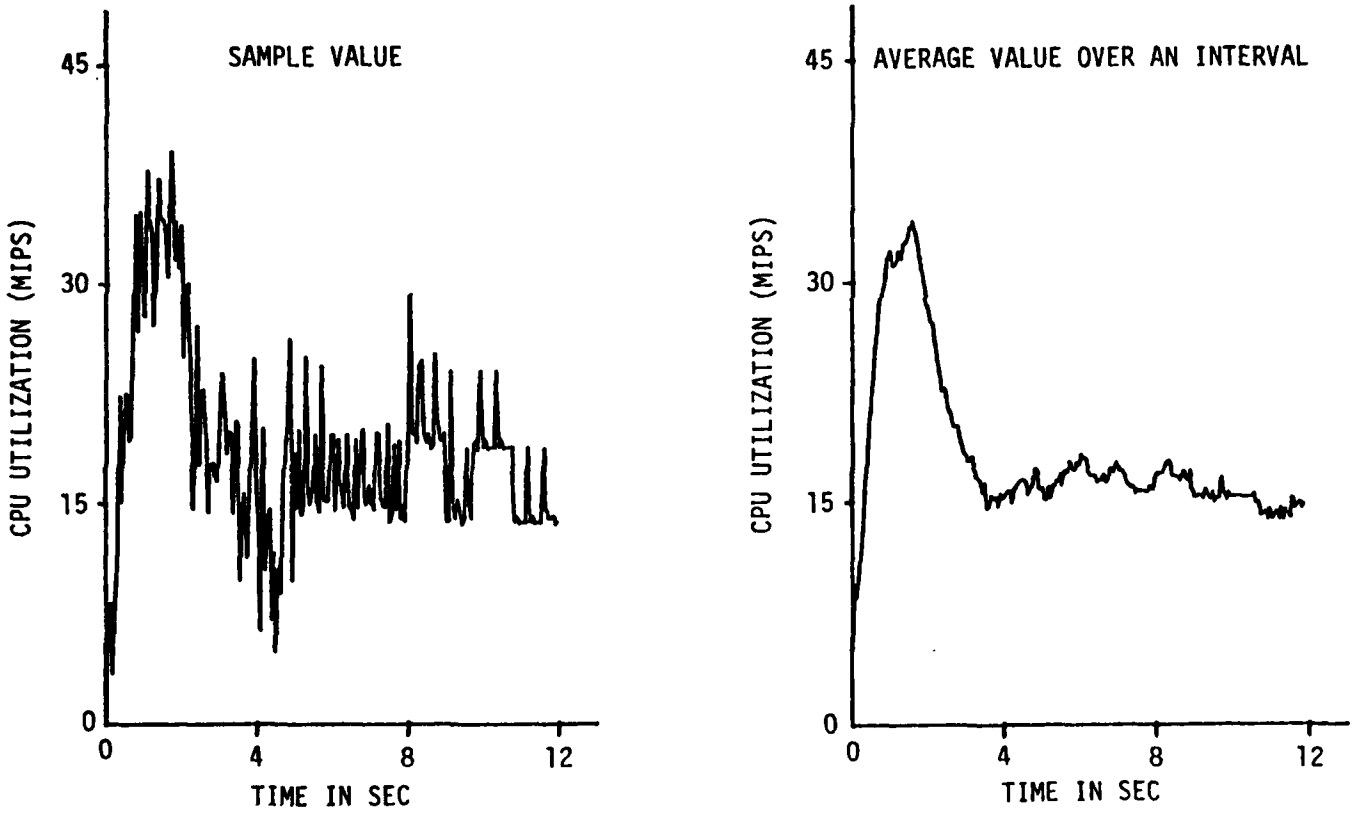


Figure C-1. Centralized Architecture, Central Node, Single-Spike Scenario

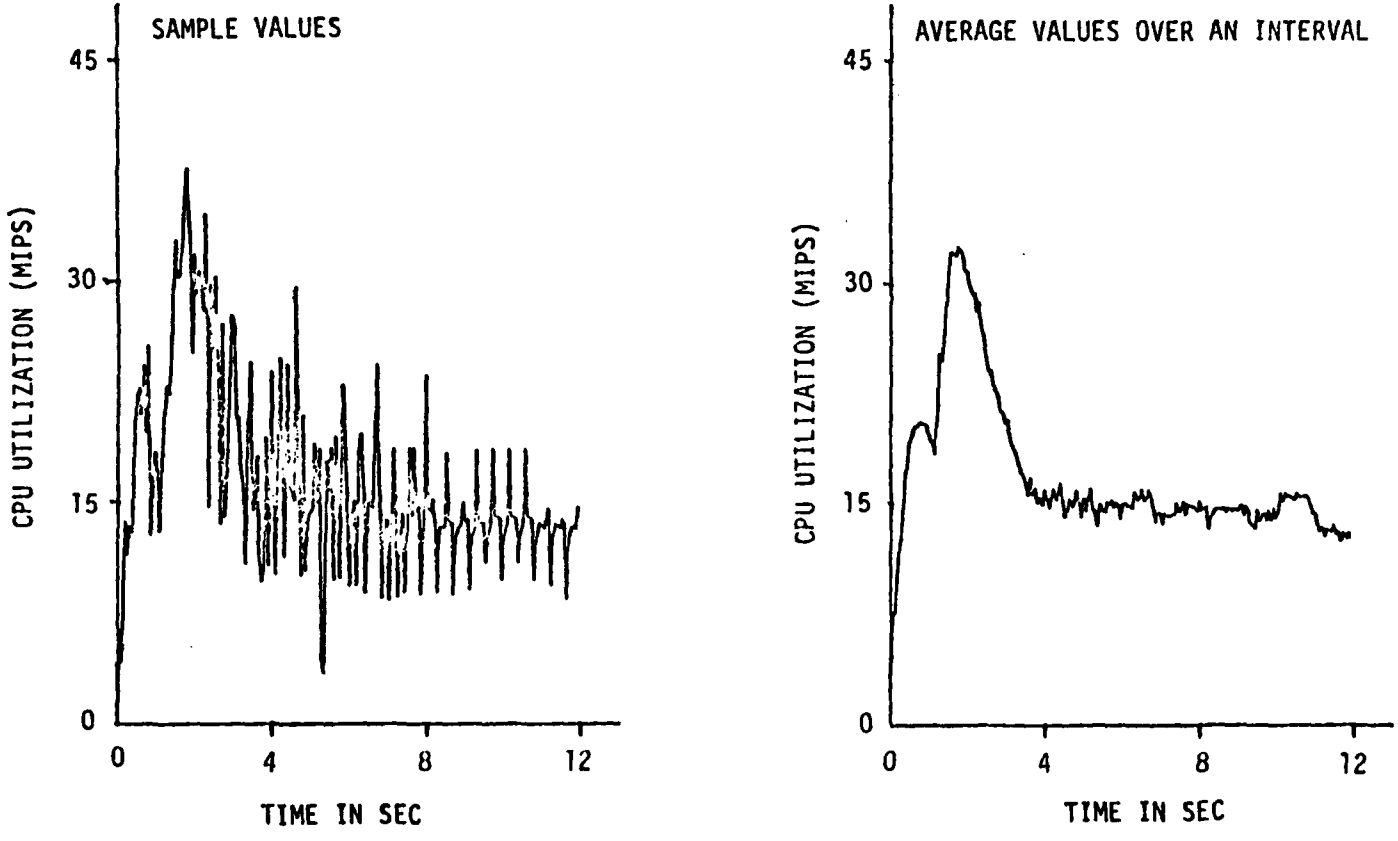


Figure C-2. Centralized Architecture, CENTRAL node, Double-Sail Scenario

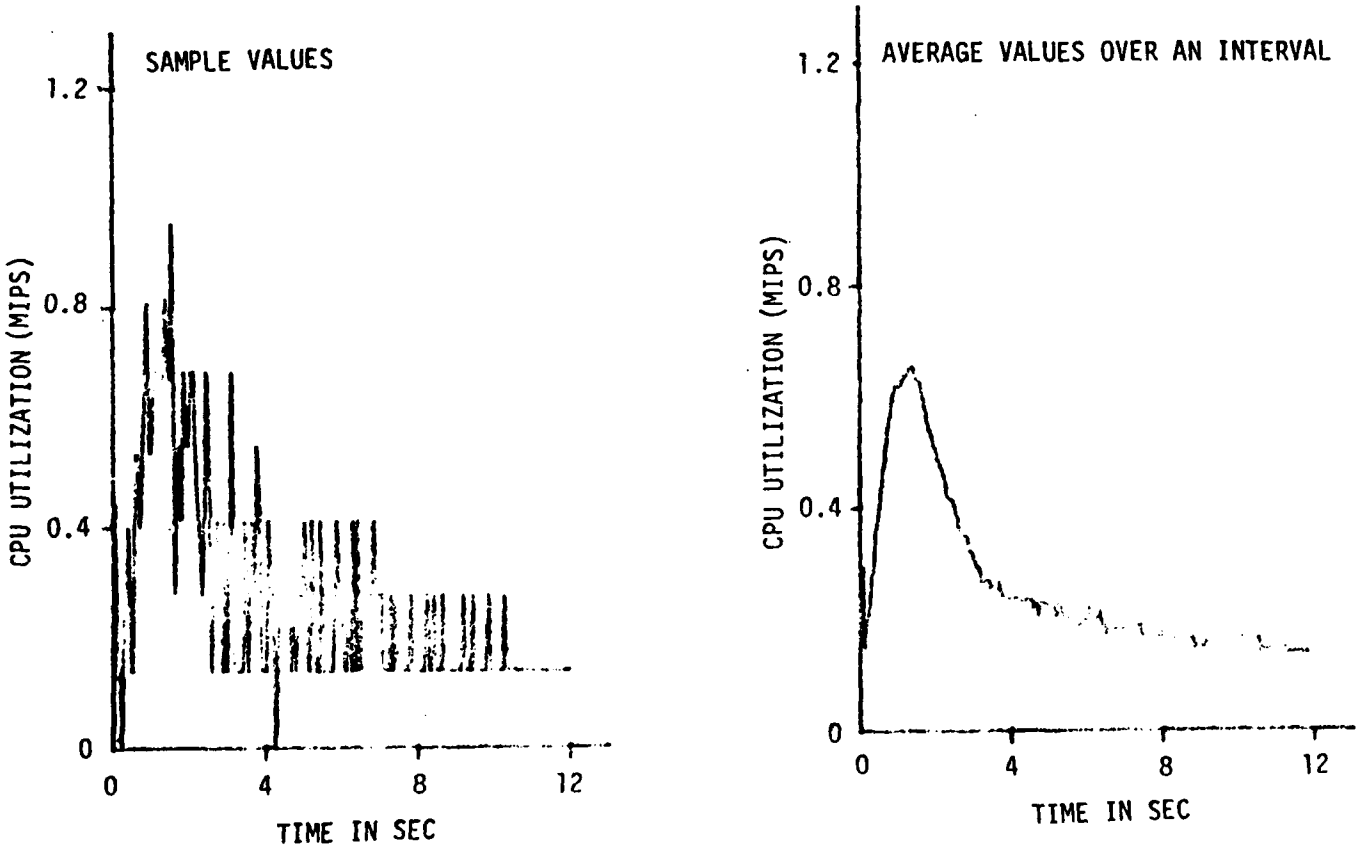


Figure C-3. Thread Architecture, RRA Node, Single-Spike Scenario

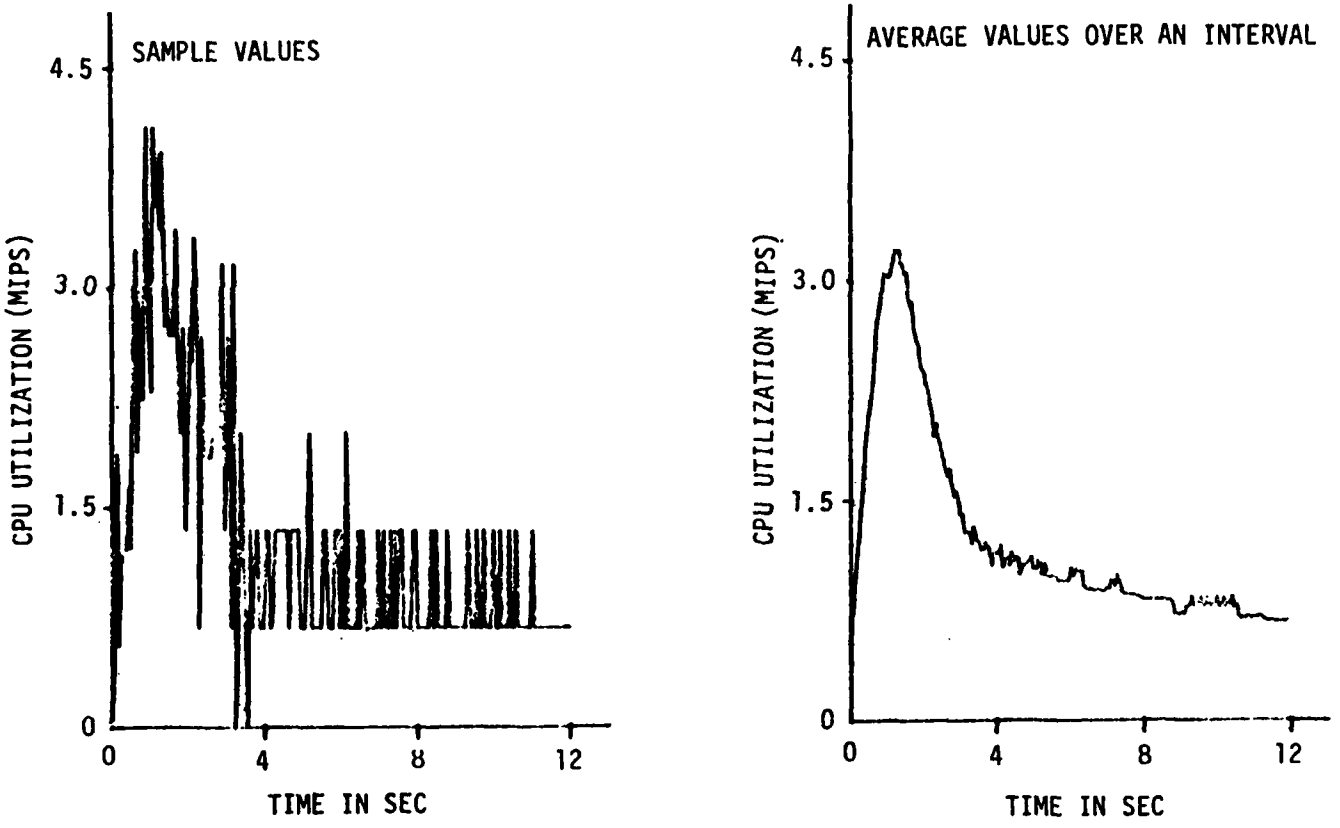


Figure C-4. Thread Architecture, MACRO Node, Single-Spike Scenario

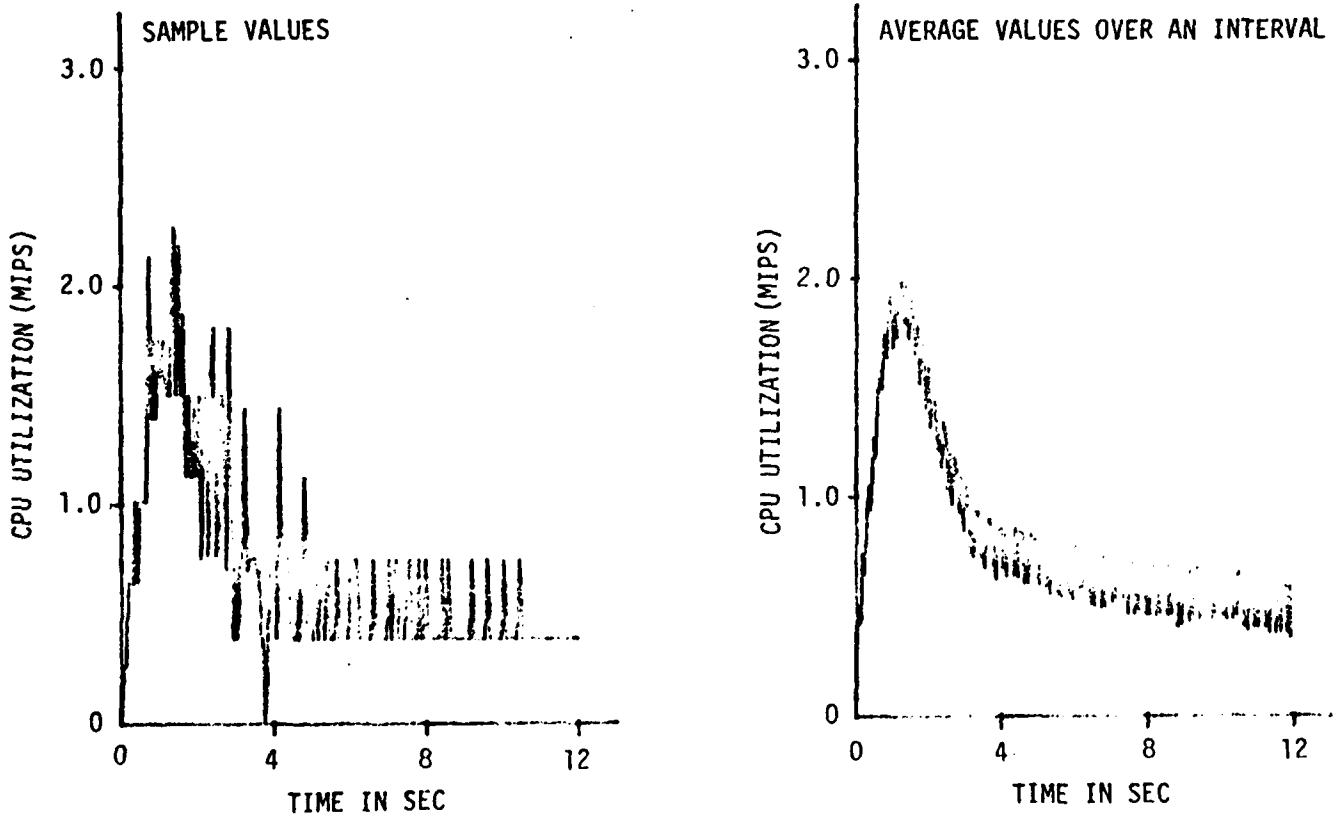


Figure C-5. Thread Architecture, MICRO Node, Single-Spike Scenario

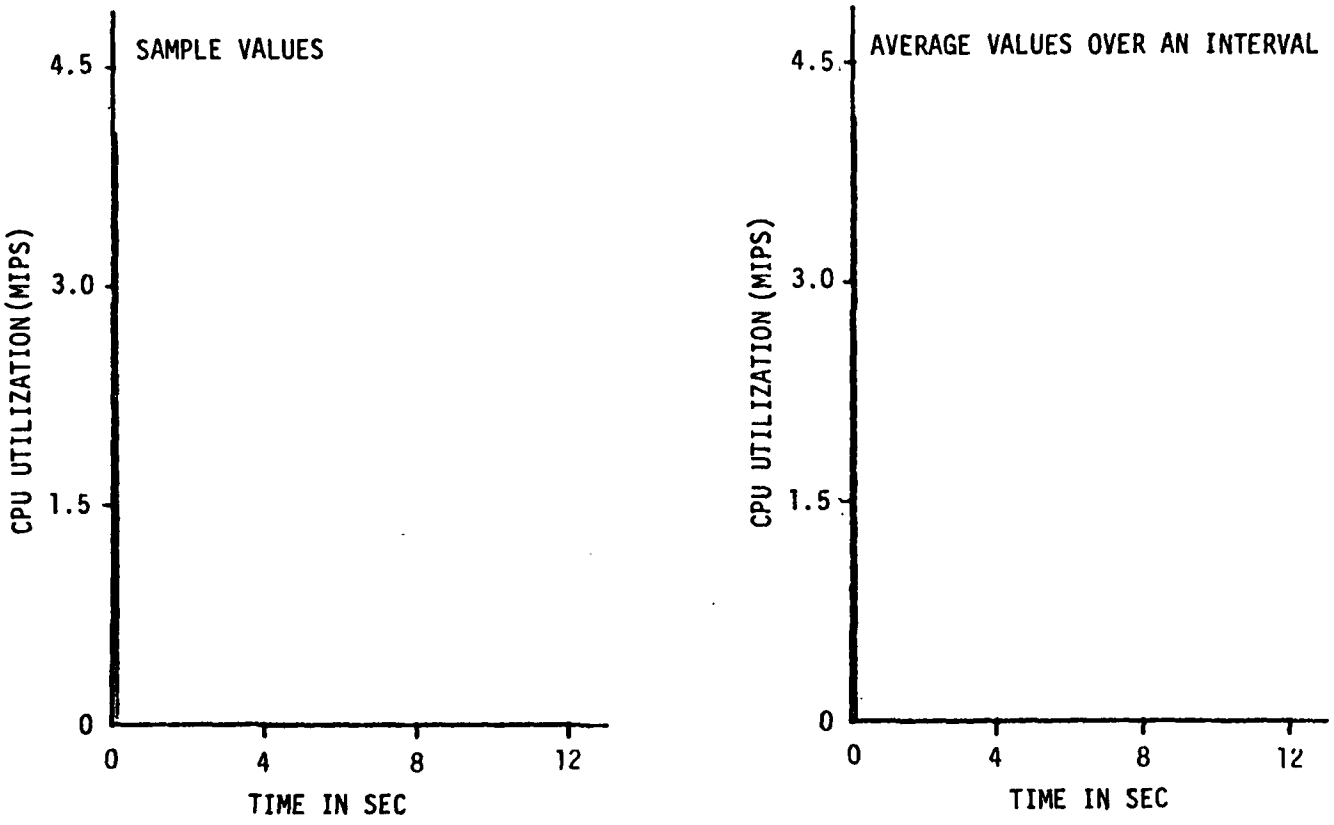


Figure C-6. Thread Architecture, SV node, Single-Spike Scenario

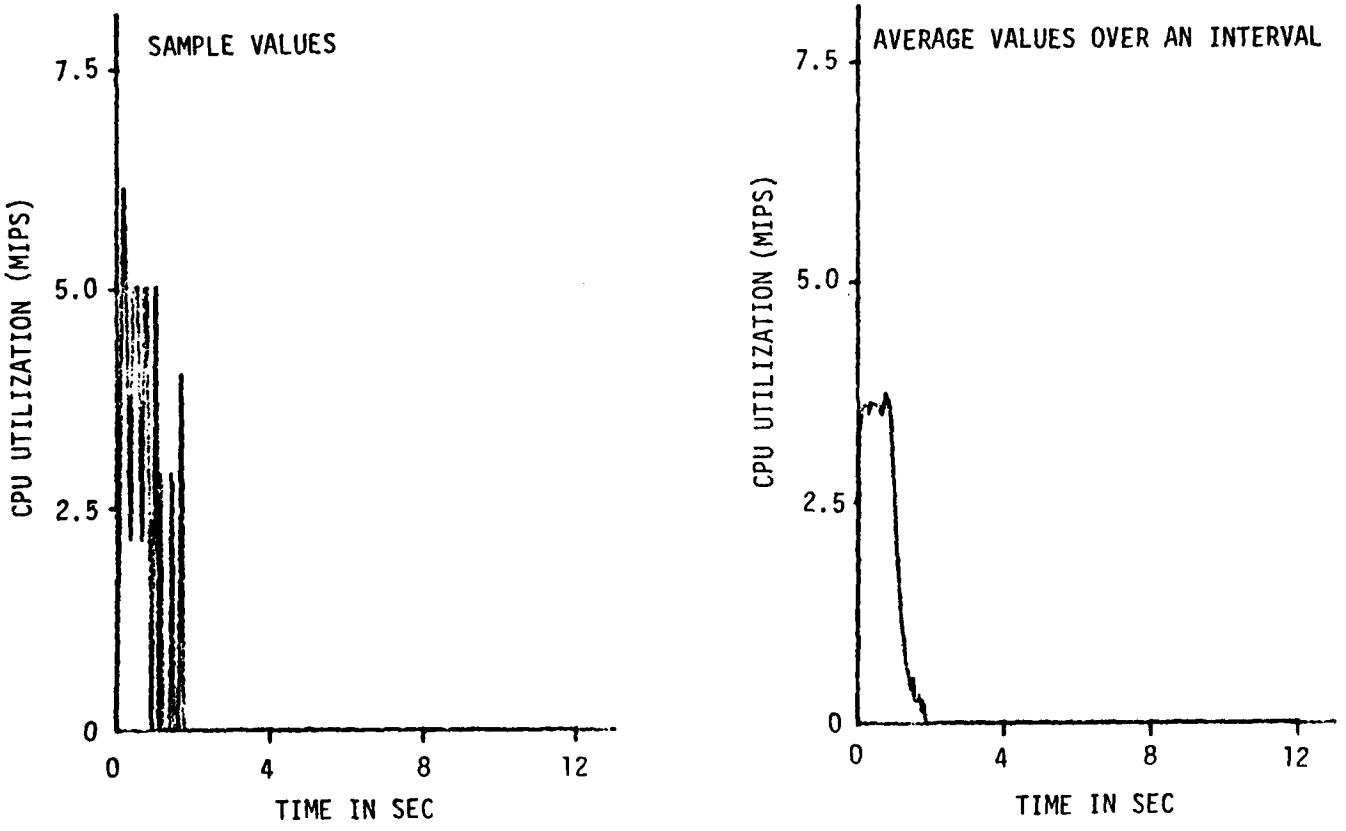


Figure C-7. Thread Architecture, TI node, Single-Spike Scenario

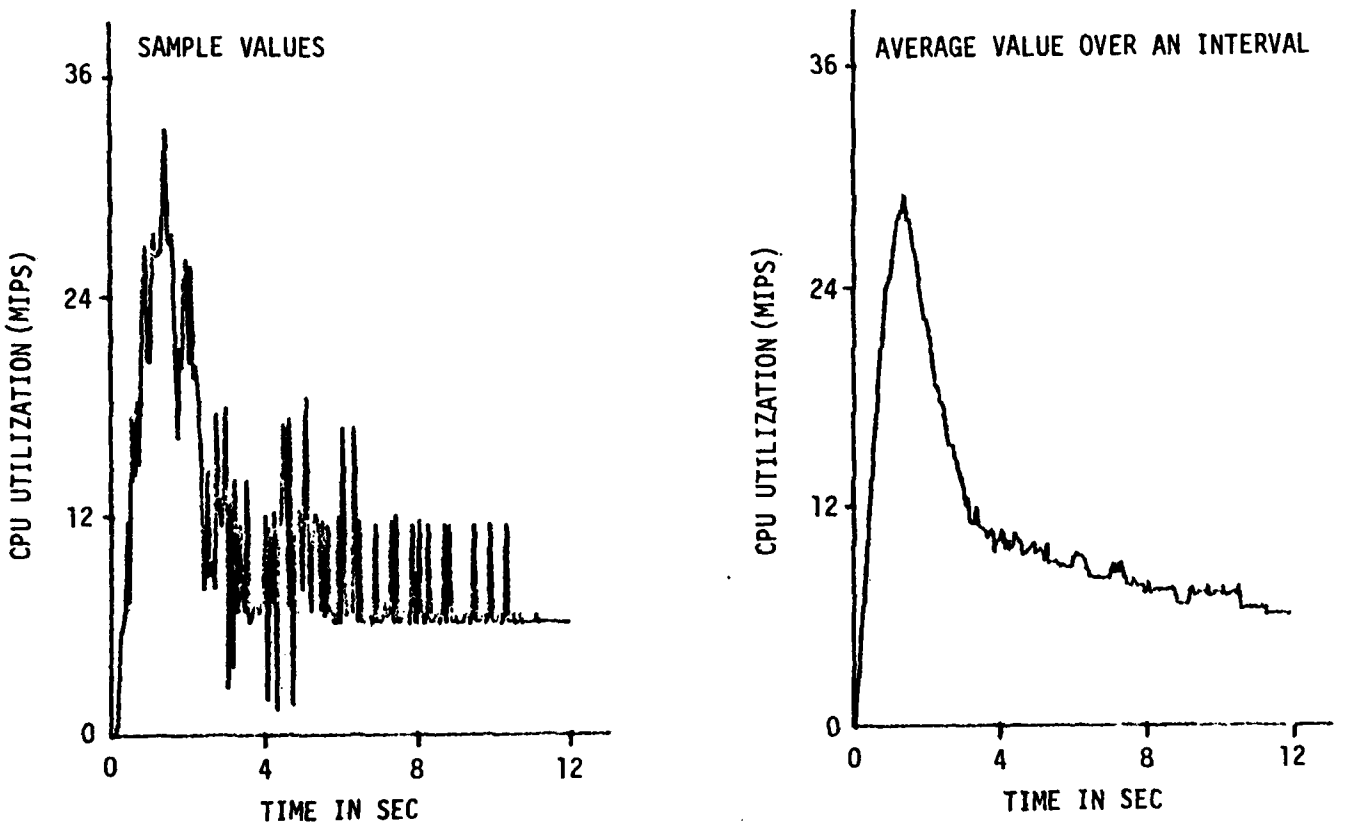


Figure C-8. Thread Architecture, OTOD Node, Single-Spike Scenario

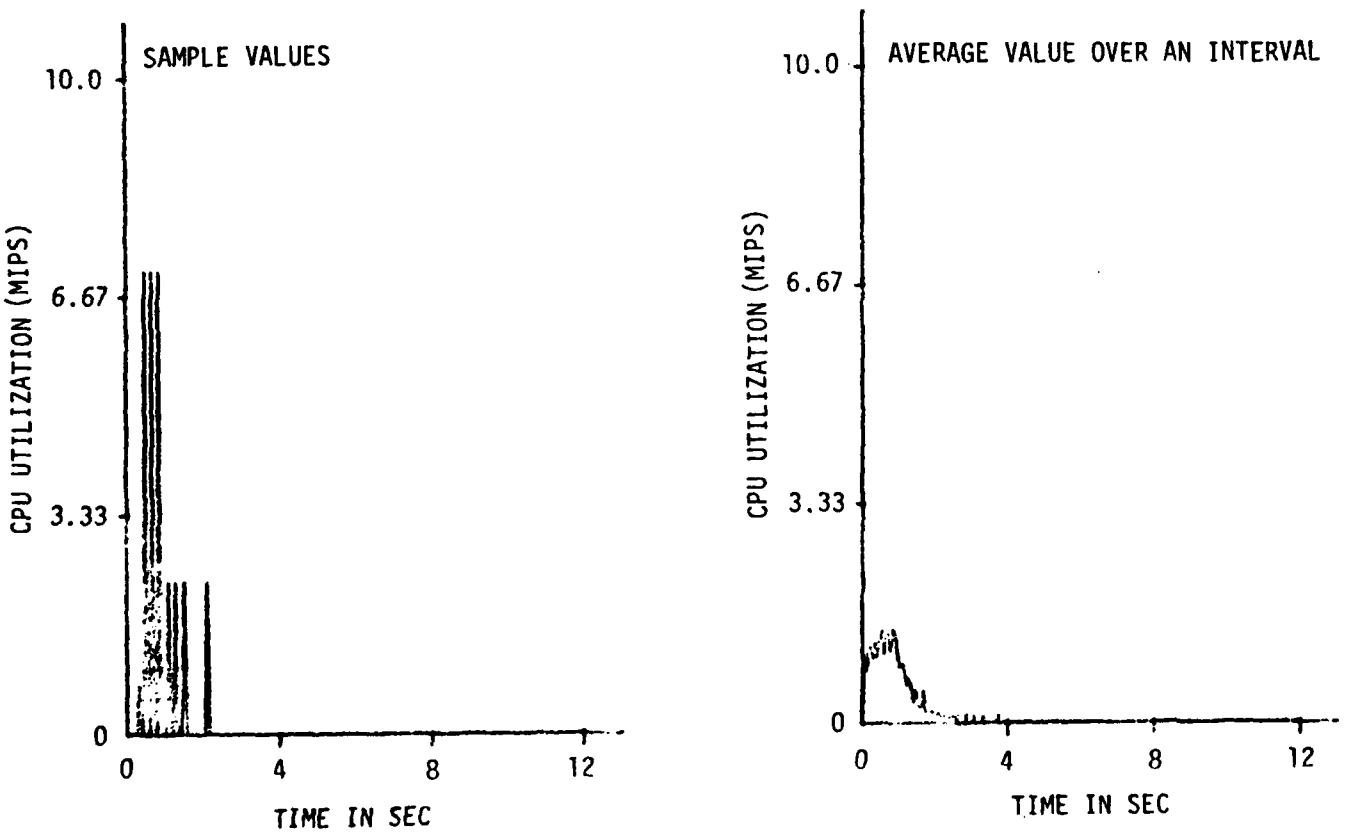


Figure C-9. Thread Architecture, REDUND node, Single-Spike Scenario

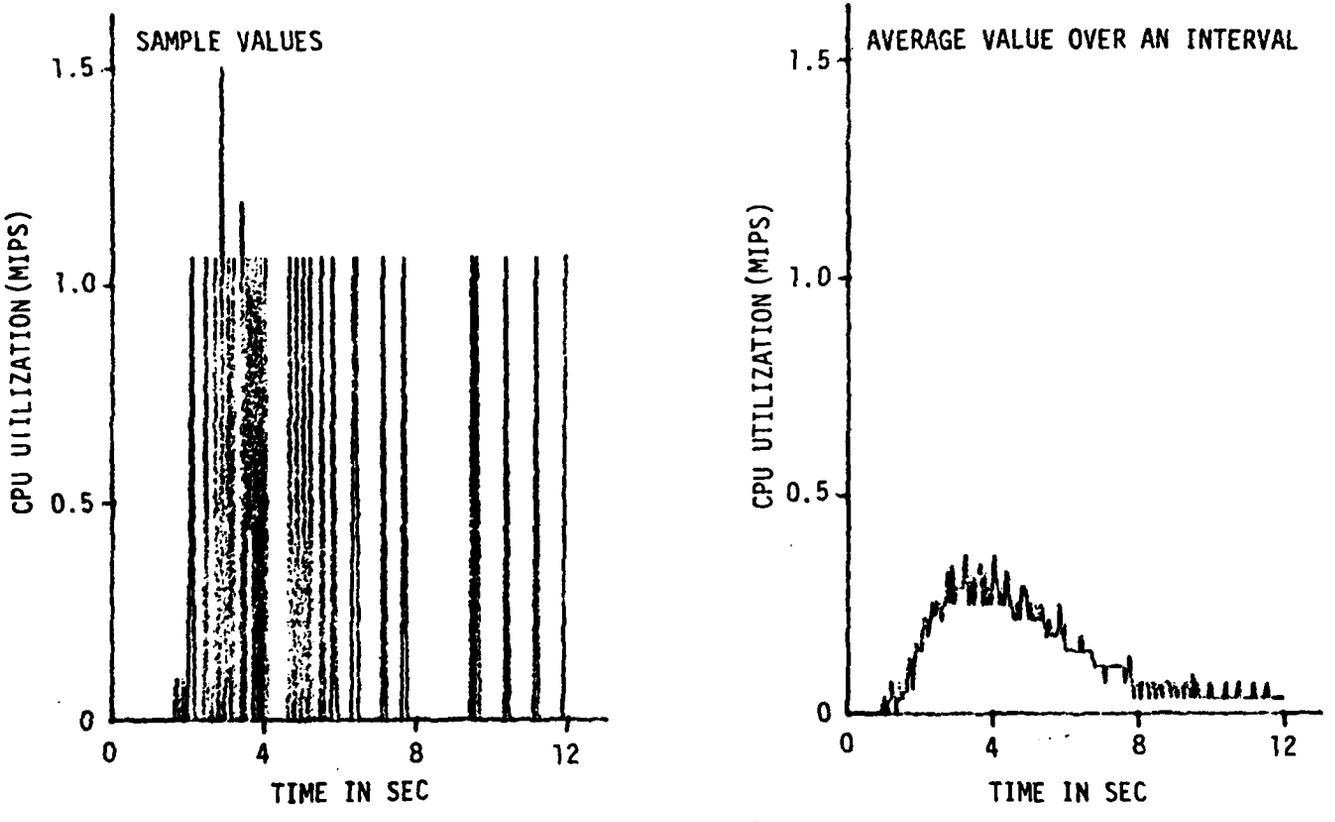


Figure C-10. Thread Architecture, ID node, Single-Spike Scenario

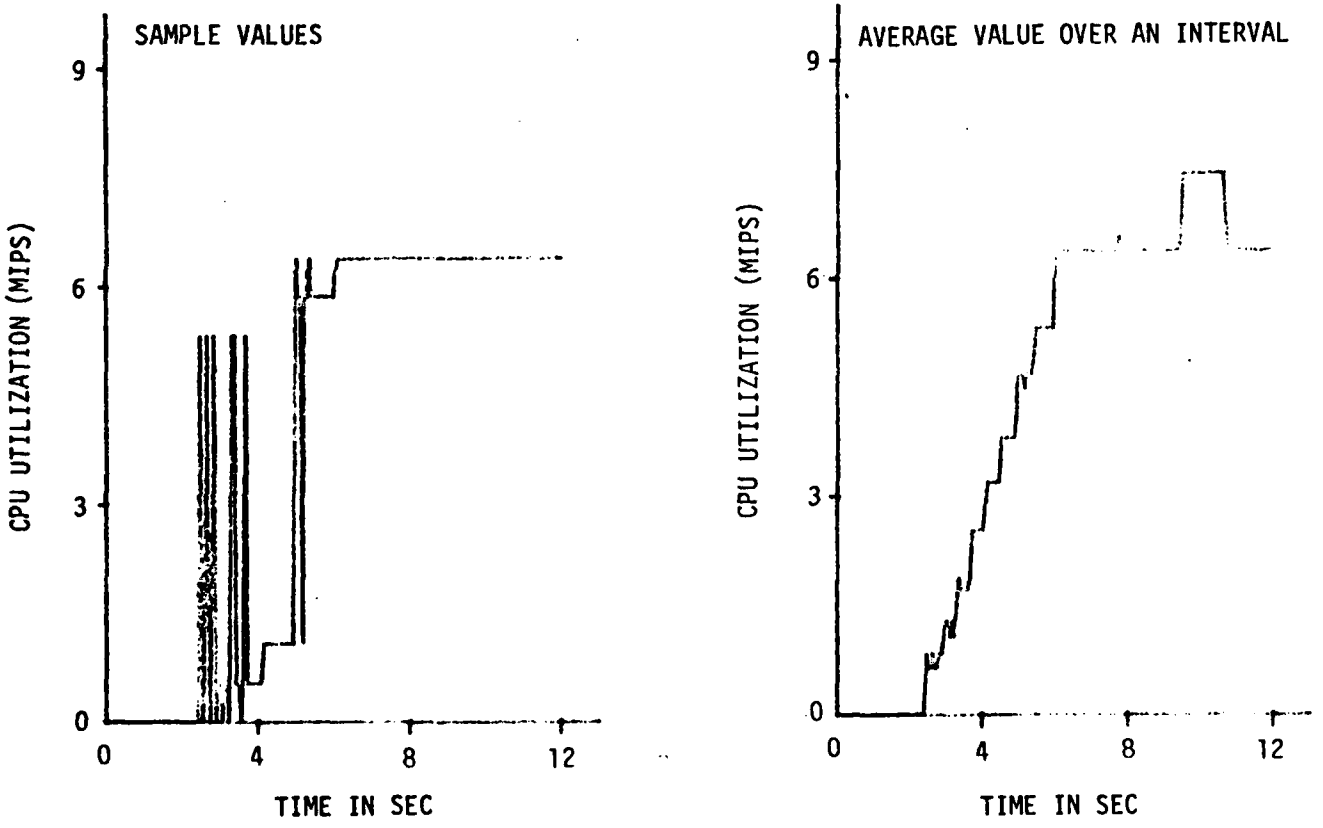


Figure C-11. Thread Architecture, IC Node, Single-Spike Scenario

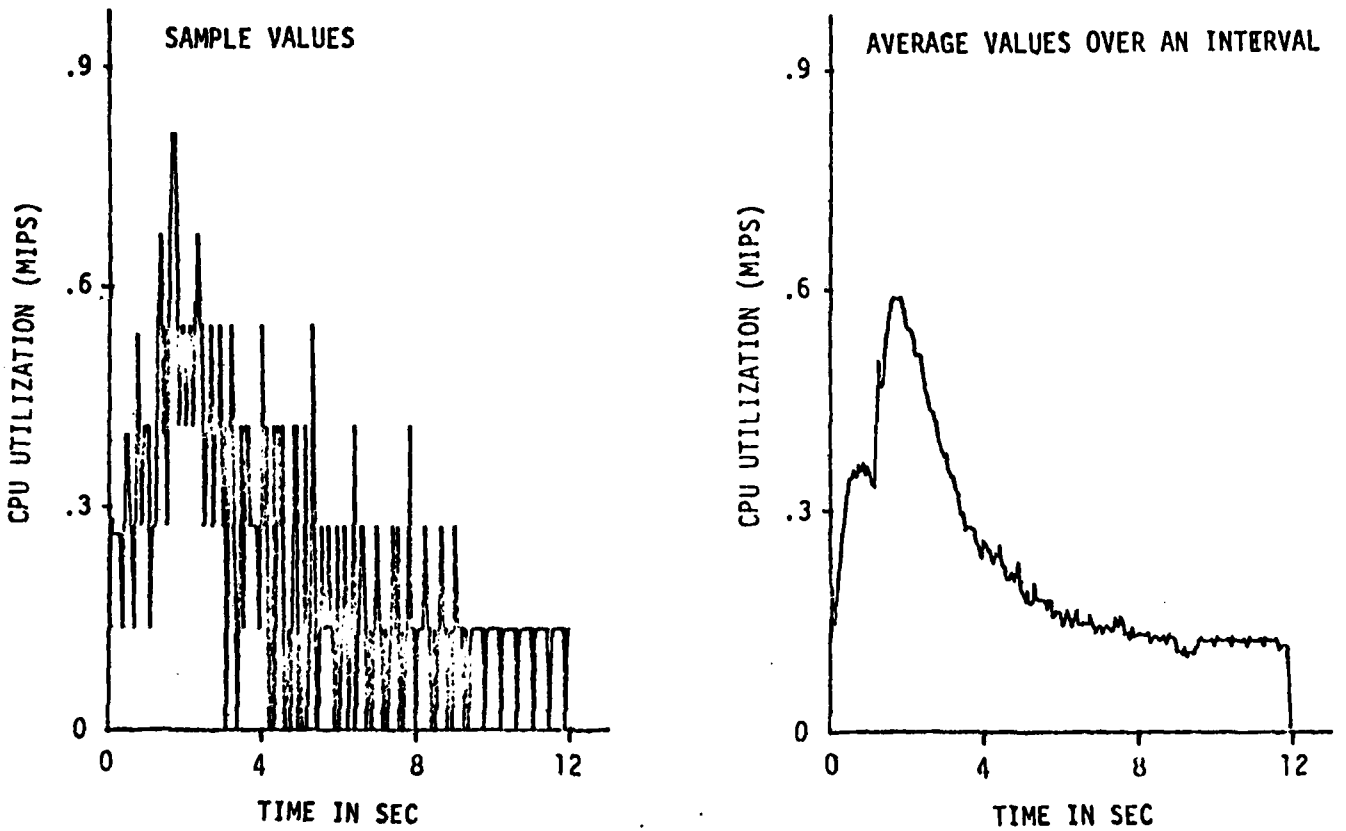


Figure C-12. Thread Architecture, RRA node, Double-Spi' Scenario

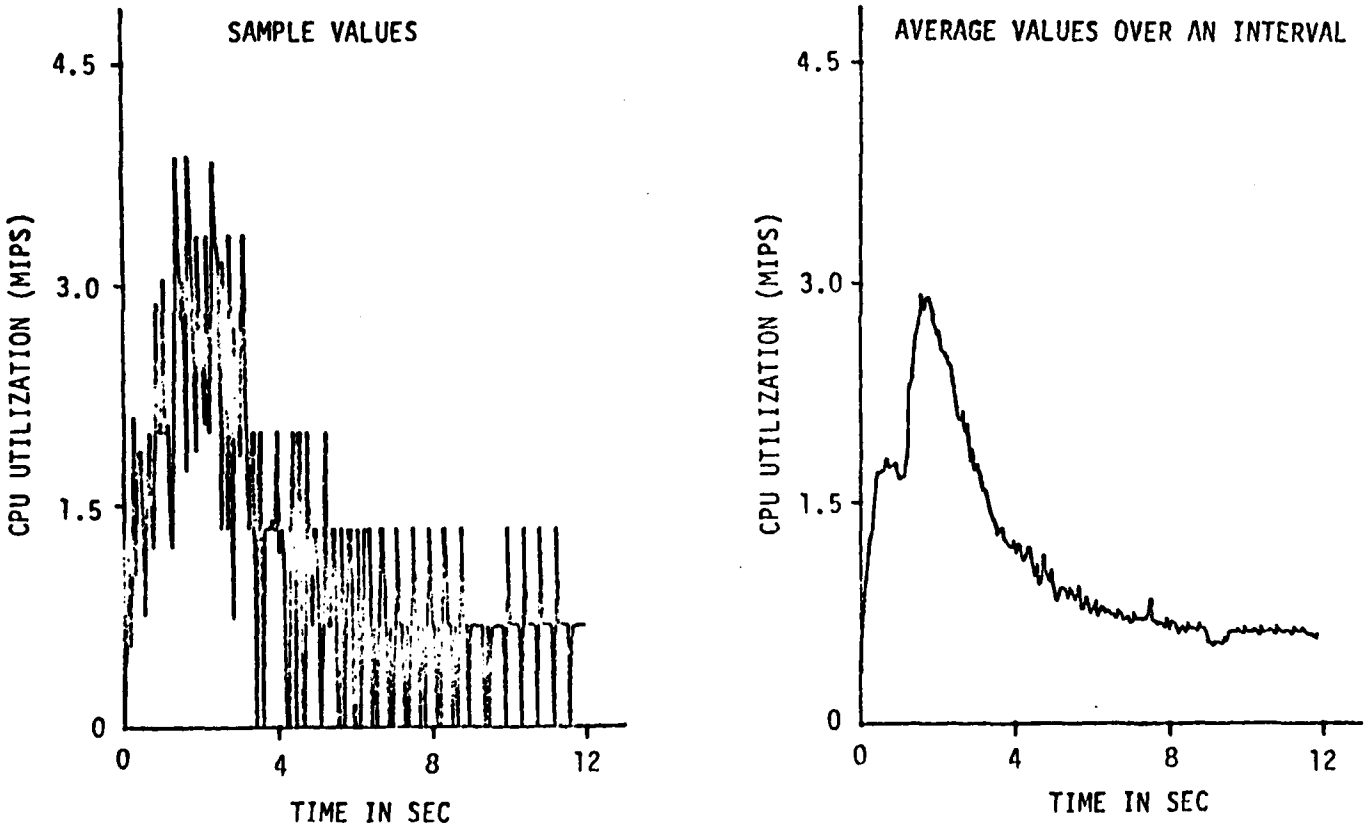


Figure C-13. Thread Architecture, MACRO node,
Double-Spike Scenario

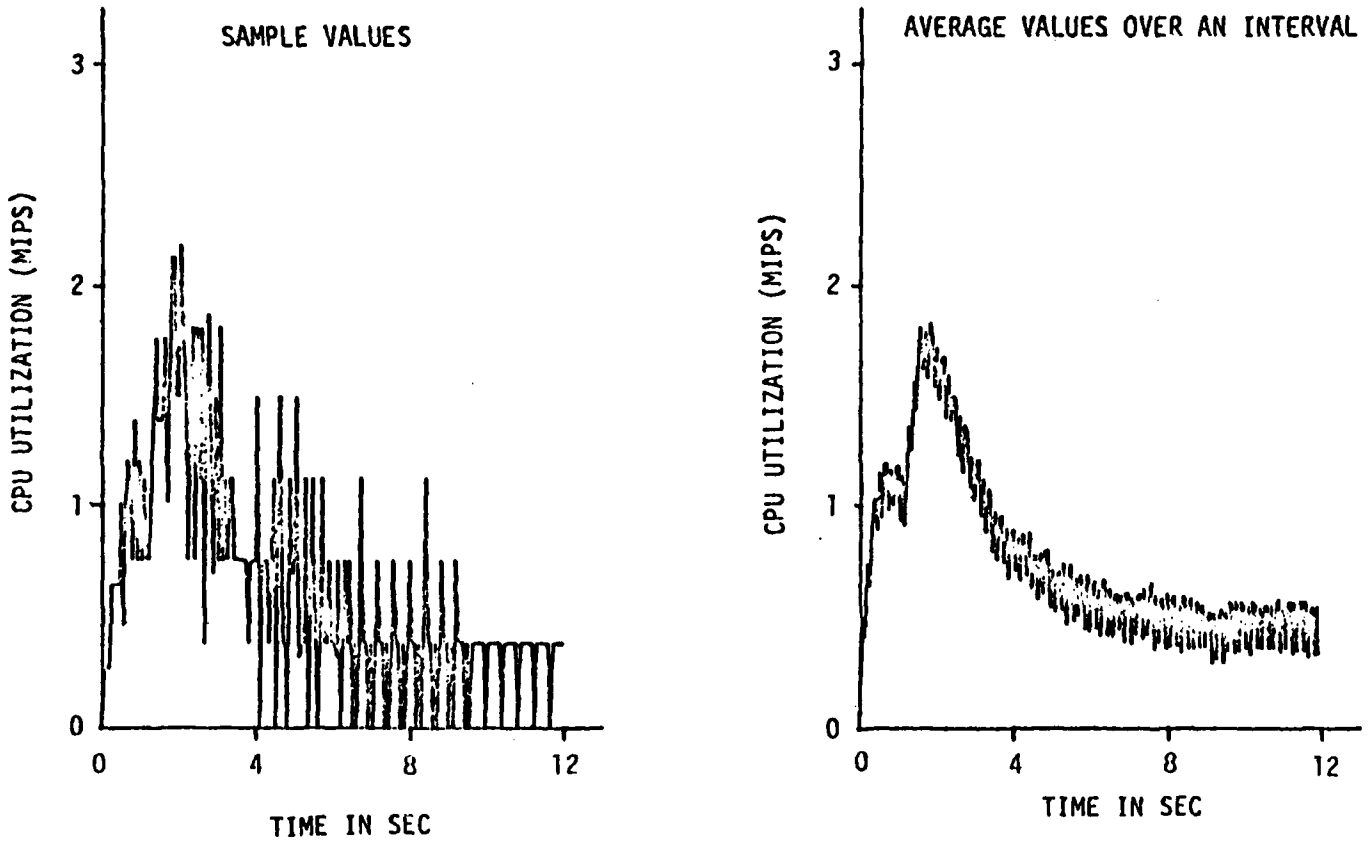


Figure C-14. Thread Architecture, MICRO node, Double-Spike Scenario

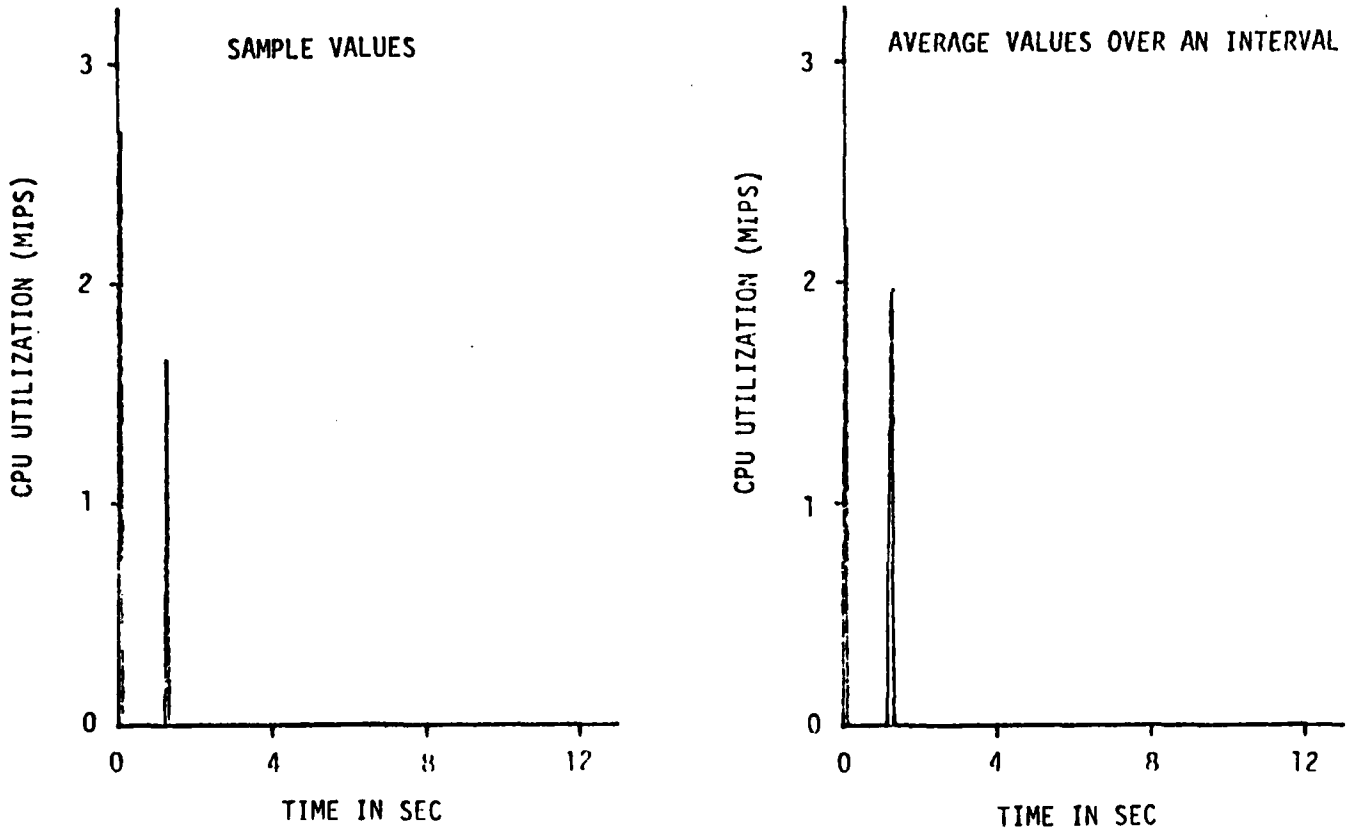


Figure C-15. Thread Architecture, SV node,
Double-Spike Scenario

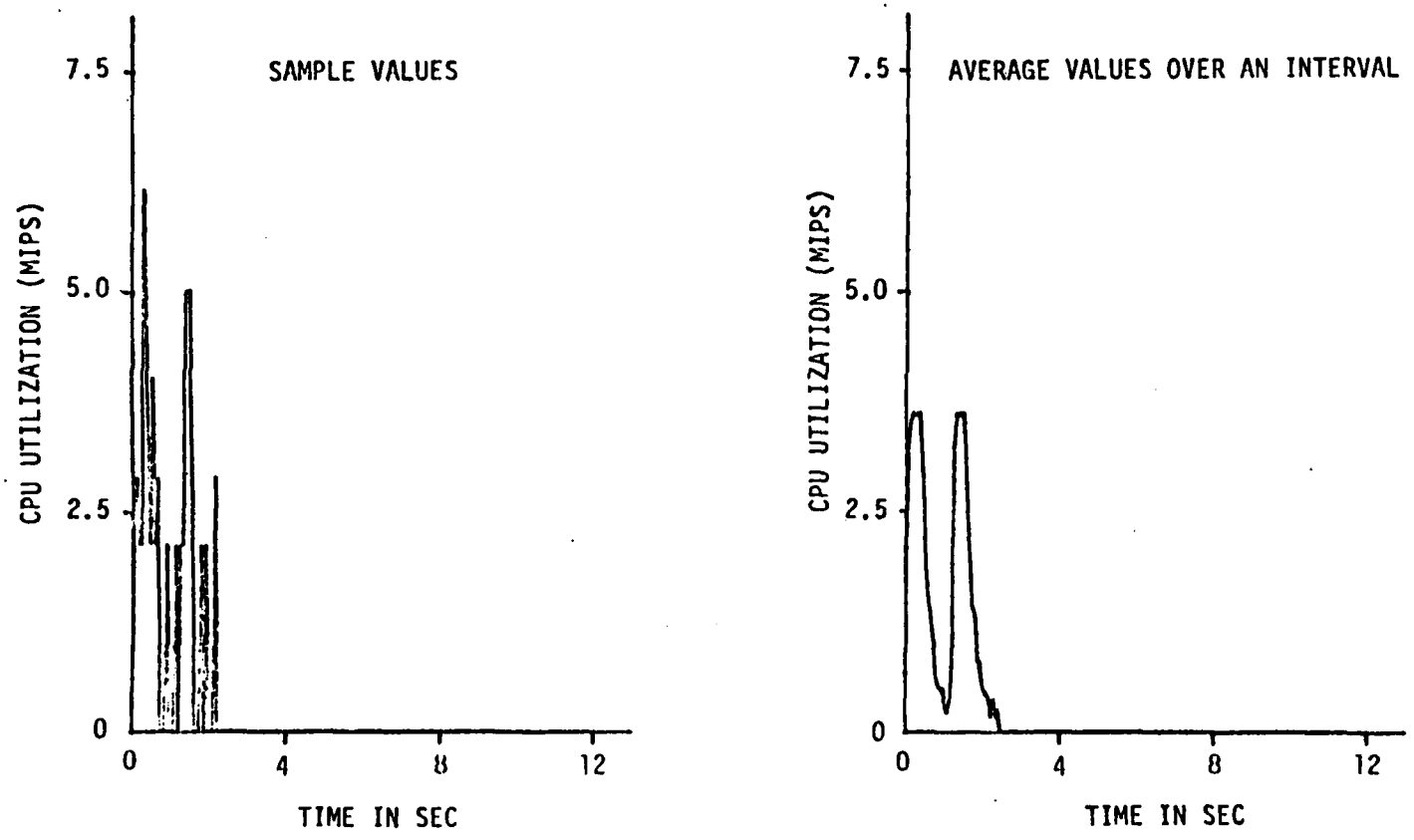


Figure C-16. Thread Architecture, TI node, Double-Spike Scenario

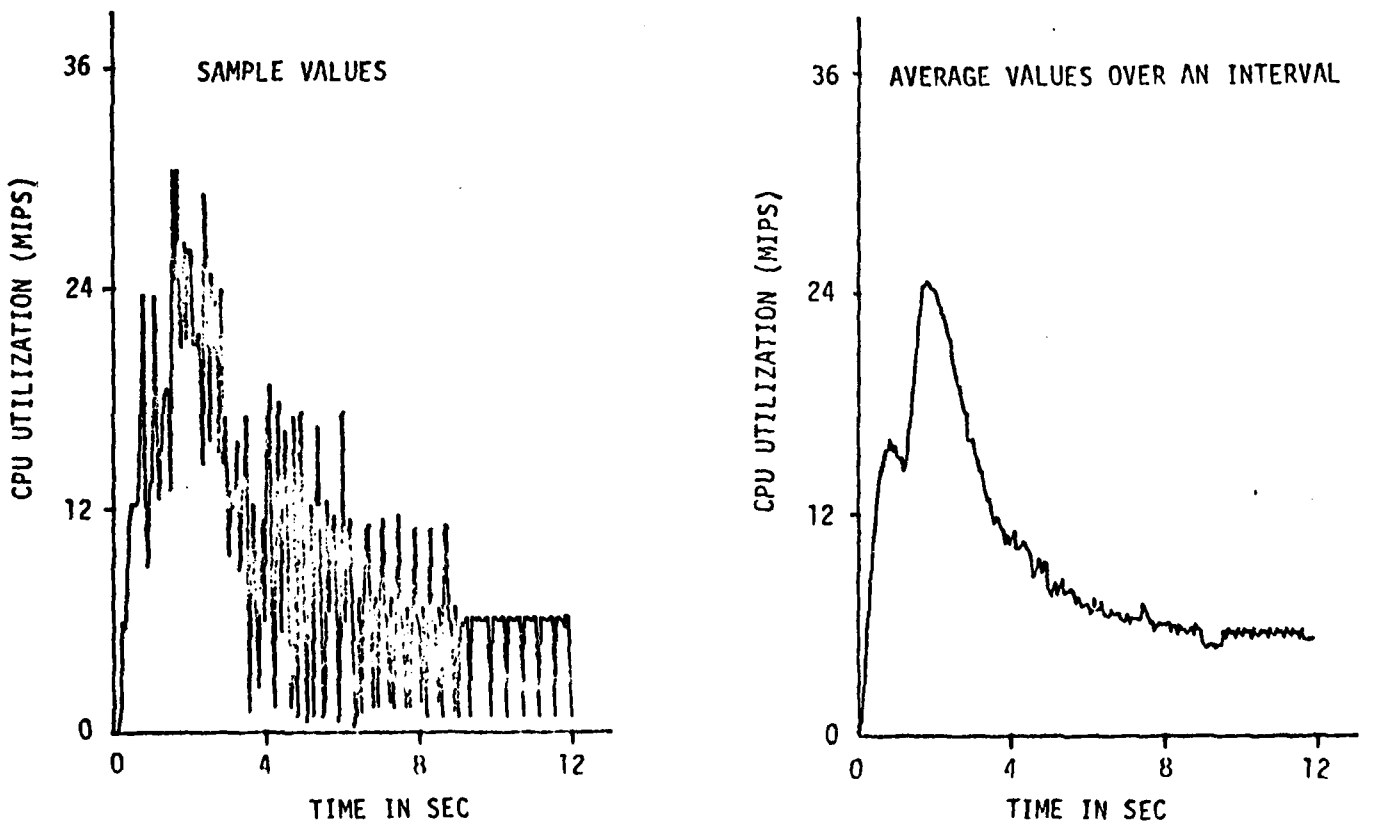


Figure C-17. Thread Architecture, OTOD node, Double-Spike Scenario

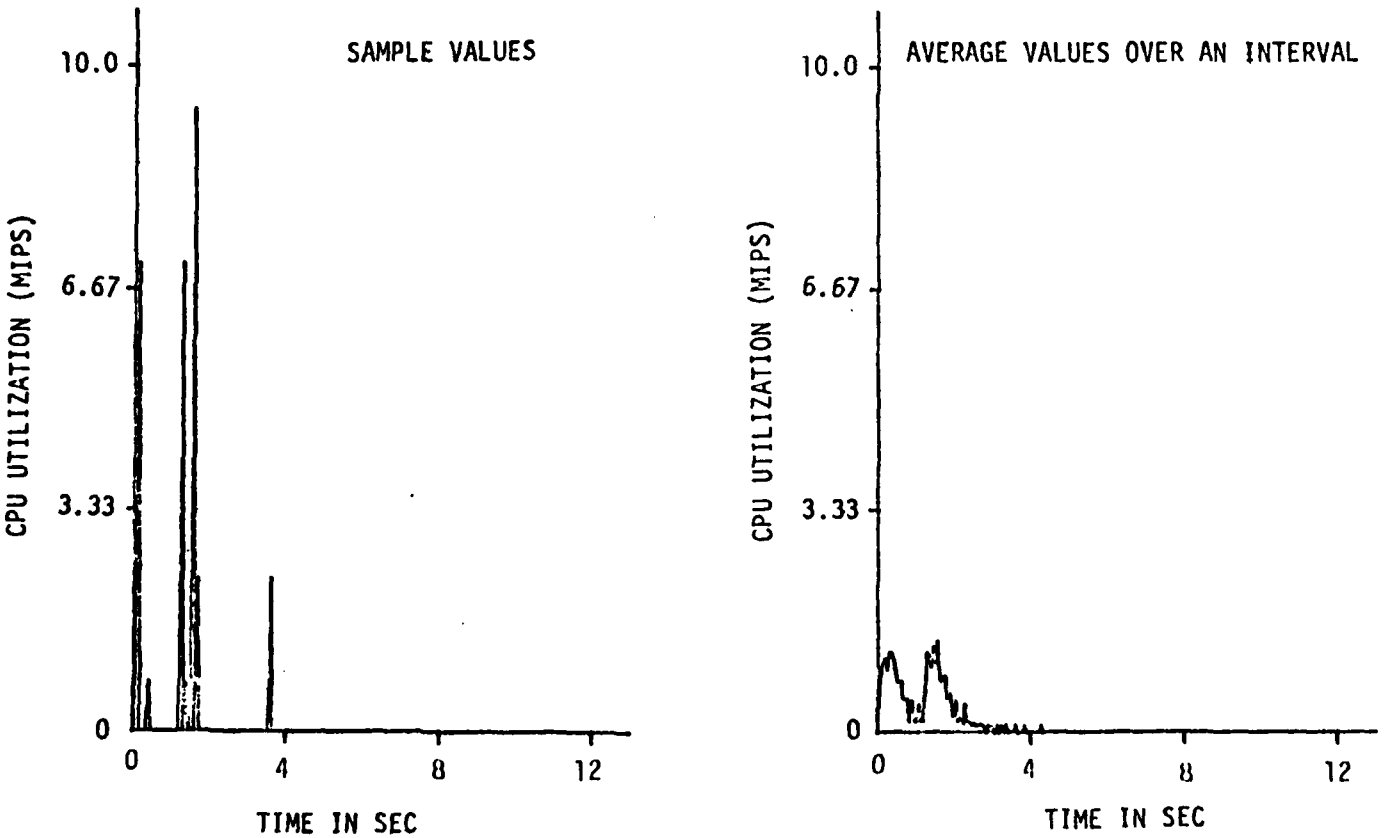


Figure C-18. Thread Architecture, REDUND node, Double-Spike Scenario

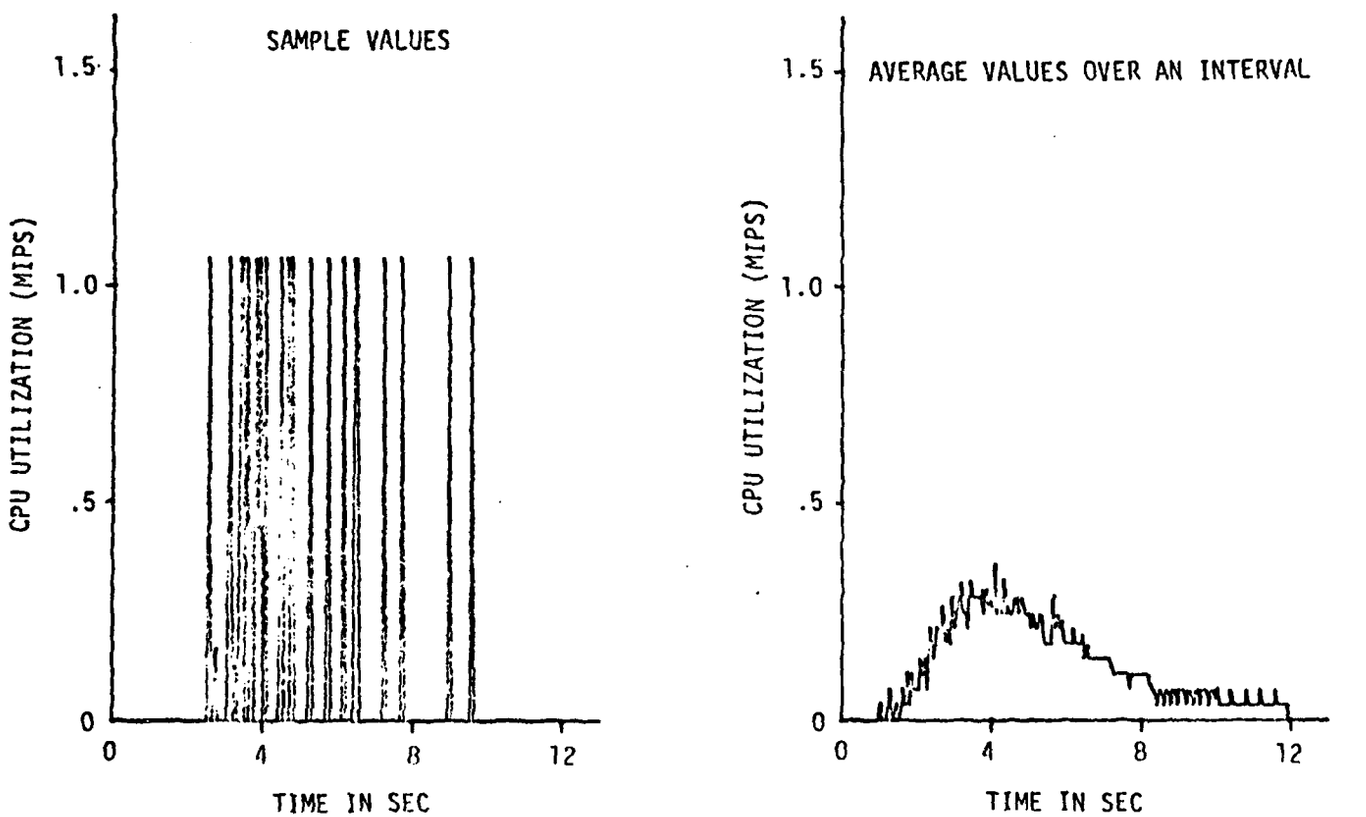


Figure C-19. Thread Architecture, IP Node, Double-Spike Scenario

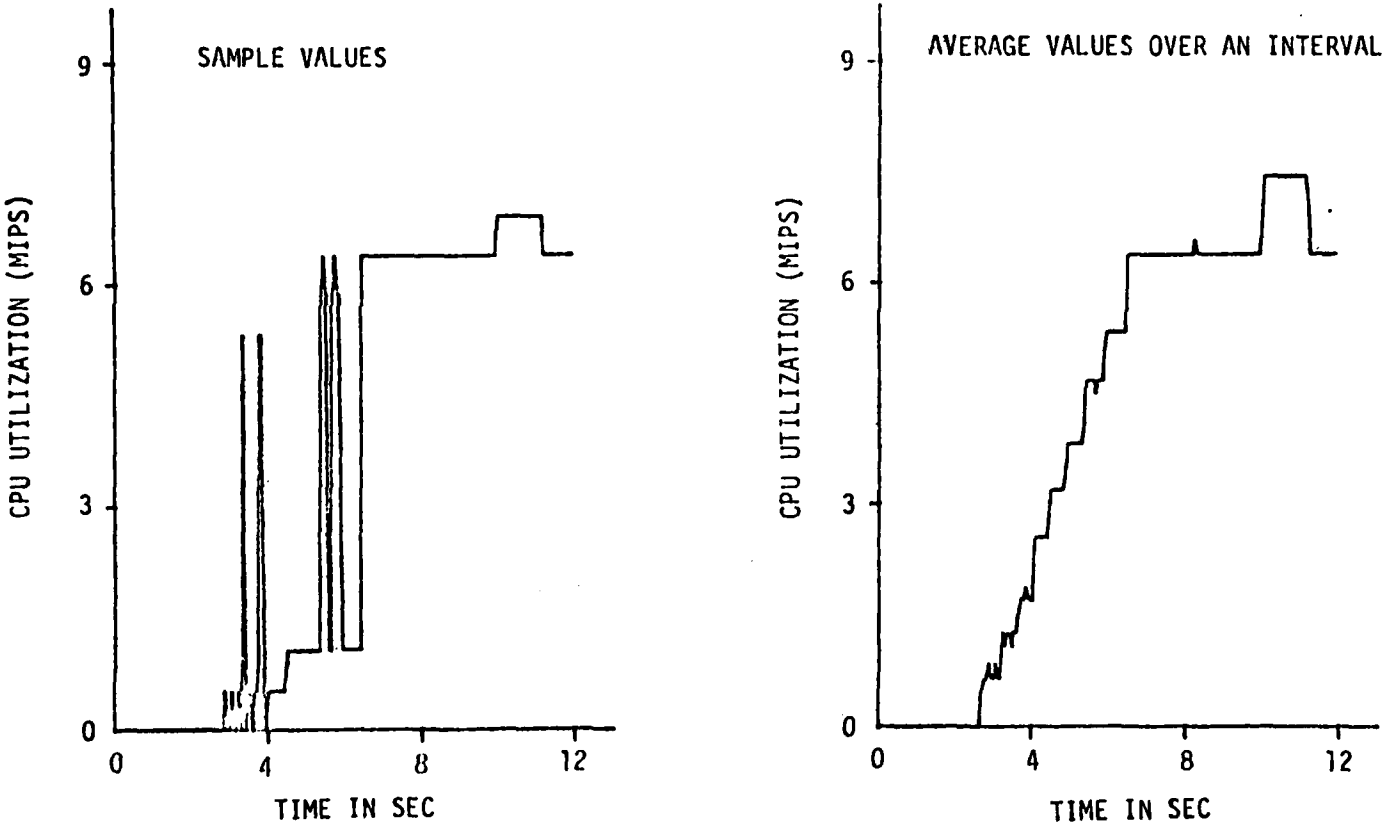


Figure C-20. Thread Architecture, IC node, Double-Sp Scenario

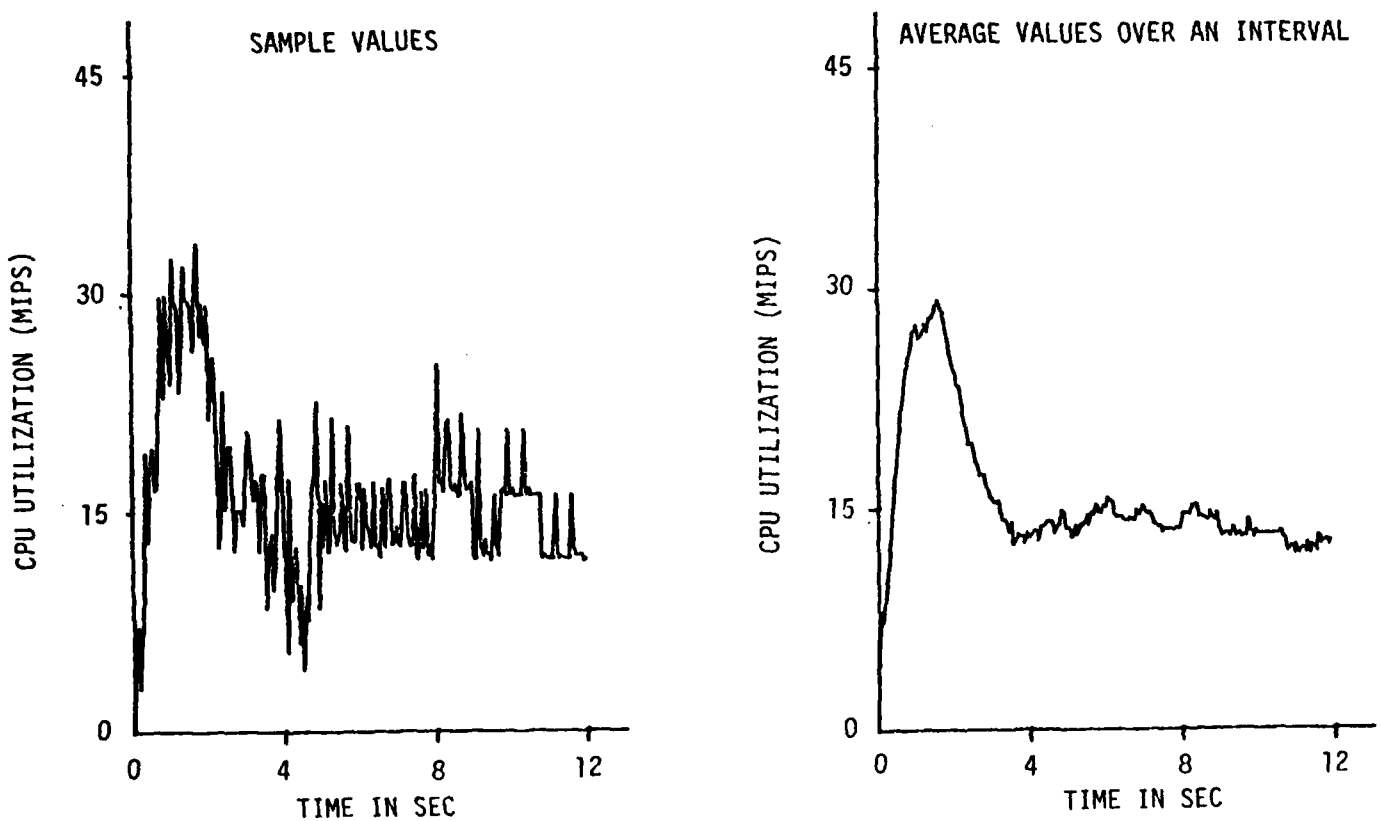


Figure C-21. Hybrid Architecture, AGGPRO Node, Single-Spike Scenario

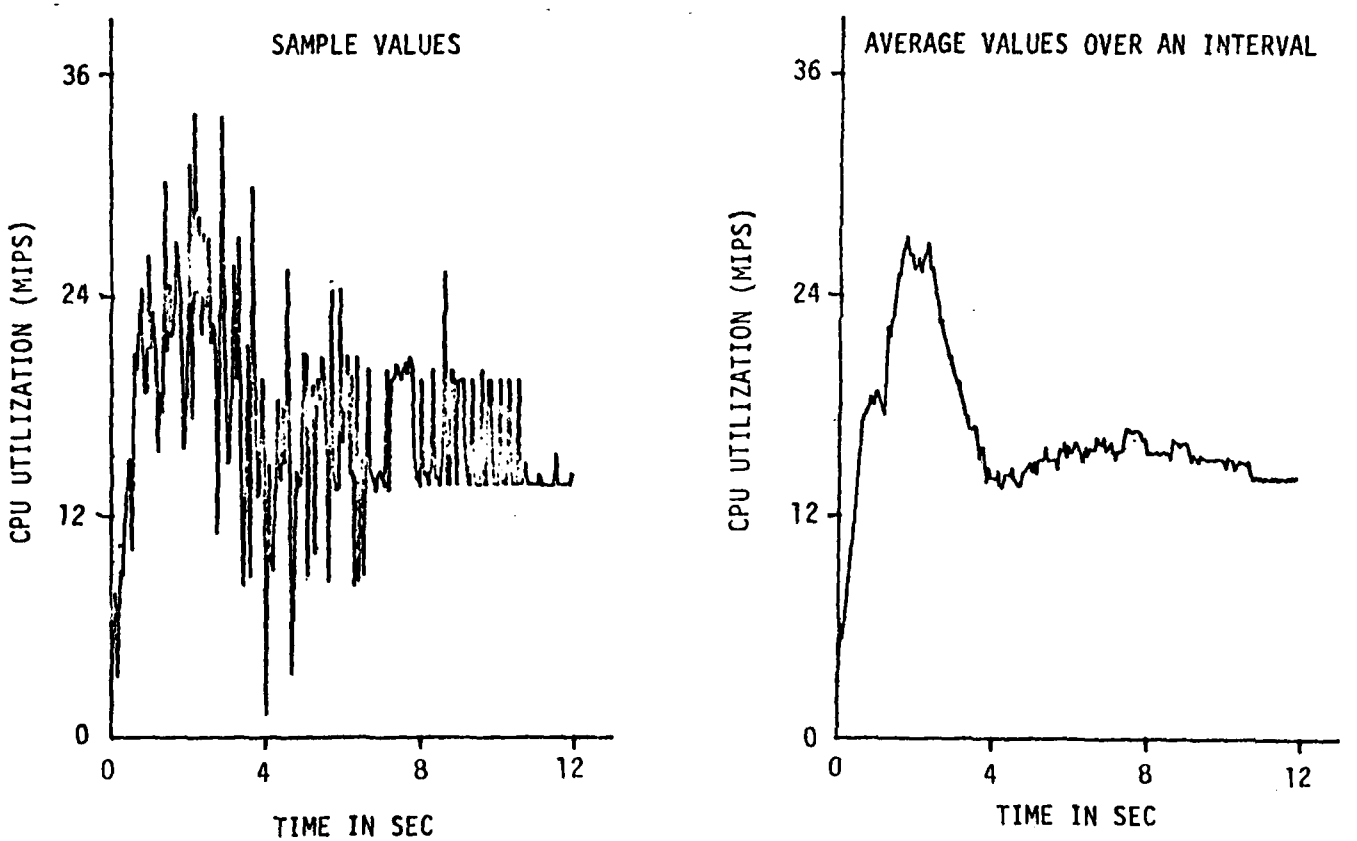


Figure C-22. Hybrid Architecture, AGGPRO node, *Double-Spike Scenario*

Appendix D
Cost Summary

VERAC No. 80-22

BMDSC

Contract No. DASG 60-80-C-0017

Recap as Invoiced

	Hours	Cost	Total Billed
Direct Labor:			
MTS III	1527		
MTS I	551		
Subtotal:			$37,209.00
ODC's:			
Materials		7323	
Travel		2454	
Subtotal:			$ 9,777.00
Overhead:			$44,651.00
Total Direct Costs and Overhead:			$91,637.00
Fee:			$ 8,247.00
Total Invoiced:			$99,884.00

Recap as Contracted

	Hours	Rate	Total Cost
Direct Labor:			
MTS III	1527	$55.48	$84,719.00
MTS I	551	$27.52	$15,165.00
Total:			$99,884.00


DAT
FILM